Bayesian Neural Network Series Post 1: Need for Bayesian Neural Networks

Kumar Shridhar
Jan 2 · 6 min read ★

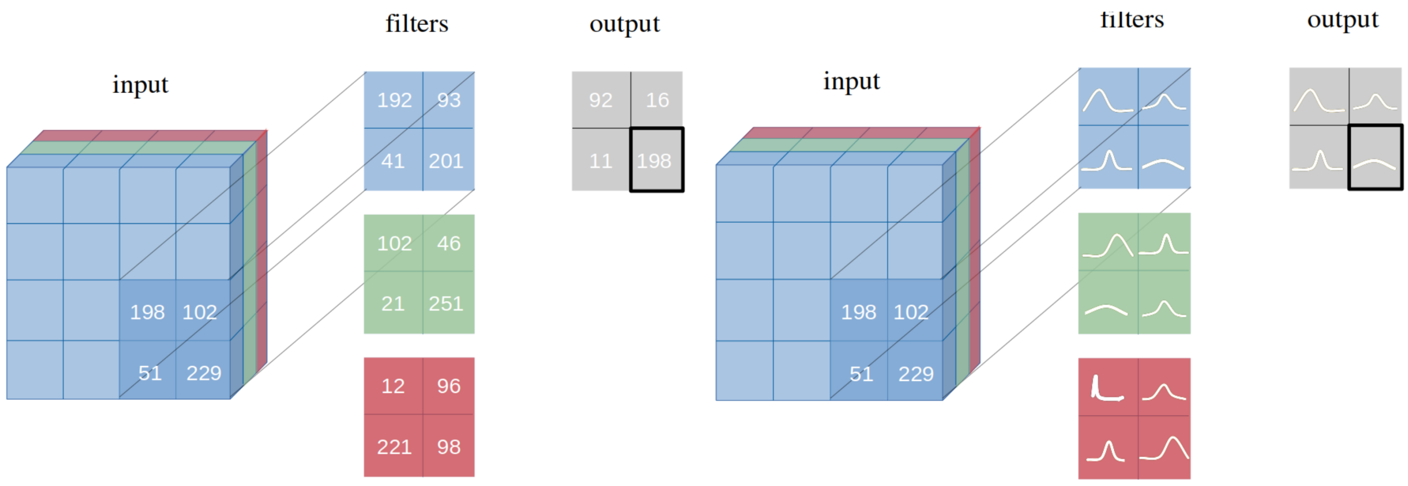

Figure 1: Network with point-estimates as weights vs Network with probability distribution as weights. Source

This post is the first post in an **eight-post series of Bayesian Convolutional Networks**. The posts will be structured as follows:

1. **Need** for Bayesian Neural Networks
2. **Background knowledge** needed to understand Bayesian Neural Networks better
3. Some **recent work** in the field of Bayesian Neural Networks
4. Bayesian Convolutional Neural Networks **using Variational Inference**
5. Build your own **Bayesian Convolutional Neural Network in PyTorch**
6. **Uncertainty estimation** in a Bayesian Neural Networks
7. **Model Pruning** in a Bayesian Neural Network

8. **Applications** in other areas (Super Resolution, GANs and so on..)

The blogs will be released every week starting first-week January 2019. Stay tuned!

. . .

Let's start this series by understanding the need for Bayesian Neural Networks in this blog.

Problem Statement

Deep Neural Networks (DNNs), are connectionist systems that learn to perform tasks by learning on examples without having prior knowledge about the tasks. They easily scale to millions of data points and yet remain tractable to optimize with stochastic gradient descent.

Convolutional Neural Networks (CNN), a variant of DNNs, have already surpassed human accuracy in the realm of image classification. Due to the capacity of CNNs to fit on a wide diversity of non-linear data points, they require a large amount of training data. This often makes CNN and neural networks in general, prone to over-fitting on datasets with small numbers of training examples per class. The model tends to fit well to the training data, but does not predict well for unseen data. This often makes neural networks incapable of correctly assessing the uncertainty in the training data and hence leads to overly confident decisions about the correct class, prediction or action.

> *If the terms in this blog seem to a bit advanced for you, it is recommended to refresh some Deep Learning basics from here.*

To understand this, let's think of training a binary classifier CNN on dogs and cats image classes. Now, when an image of a leopard is encountered in the test dataset, the model should ideally predict it neither dog nor cat (a 50 per cent probability for dog and 50 per cent probability for cat class). However, due to a softmax function at the output layer to achieve the probability score, it squishes one class output probability score and maximizes the other, leading to an overconfident decision for one class. This is one of the major problems of a point-estimate neural network.

> *Note that the term point-estimate is used for a neural network where weights are represented by a single point. On the other hand, a Bayesian neural network represents the*

> *weights in form of distribution as seen in Figure 1.*

But do we really need a Bayesian neural network for it? Various regularization techniques for controlling over-fitting are used in practice, namely early stopping, weight decay, L1 or L2 regularizations and currently the most popular and empirically very effective technique, dropout.

If we can solve the issue of making overconfident decisions and prevent the over-fitting of models by regularizing the model, then the question remains as it is: Why do we need a Bayesian neural network?

In short, the answer is: a measure of uncertainty in the prediction is missing from the current neural networks architectures, but Bayesian neural networks incorporate this.

. . .

Current Situation

Deep neural networks have been successfully applied to many domains, including very sensitive domains like health-care, security, fraudulent transactions and many more. These domains rely heavily on the predictions accuracy of the model and even one overconfident decision can result in a big problem. Also, these domains have very imbalanced datasets (one in a million fraudulent transactions, nearly five per cent of all tests result in positive cancer results, less than one per cent email is spam), and this leads to the model being over-fitted to the over-sampled class.

From a probability theory perspective, it is unjustifiable to use single point-estimates as weights to base any classification on.
Bayesian neural networks, on the other hand, are more robust to over-fitting, and can easily learn from small datasets. The Bayesian approach further offers uncertainty estimates via its parameters in form of probability distributions (see Figure 1). At the same time, by using a prior probability distribution to integrate out the parameters, the average is computed across many models during training, which gives a regularization effect to the network, thus preventing over-fitting.

. . .

The practicality of Bayesian neural networks

Bayesian posterior inference over the neural network parameters is a theoretically attractive method for controlling over-fitting; however, modelling a distribution over the kernels (also known as filters) of a CNN has **never been attempted successfully** before, perhaps because of the vast number of parameters and extremely large models commonly used in practical applications.

Even with a small number of parameters, **inferring model posterior in a Bayesian neural network is a difficult task**. Approximations to the model posterior are often used instead, with the variational inference being a popular approach. Here, one would model the posterior using a simple variational distribution such as a Gaussian distribution, and try to fit the distribution's parameters to be as close as possible to the true posterior. This is done by **minimising the Kullback-Leibler divergence** between this simple variational distribution and the true posterior. Many have followed this approach in the past for standard neural networks models.
But the variational approach used to approximate the posterior in Bayesian NNs can be fairly **computationally expensive** — the use of Gaussian approximating distributions increases the number of model parameters considerably, without increasing model capacity by much. Blundell et al. (2015), for example, used Gaussian distributions for Bayesian NN posterior approximation and have doubled the number of model parameters, yet report the same predictive performance as traditional approaches using dropout. This makes the approach unsuitable in practice to use with CNN as the increase in the number of parameters is too costly.

. . .

How do we do it then?

There are many ways to build the Bayesian neural networks (we will ponder over a lot of them in Blog 3). However, in this series, we will focus on building a Bayesian CNN using Bayes by Backprop. The 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 lend itself to exact integration. So, we approximate the intractable true posterior probability distributions $p(w|D)$ with variational probability distributions $q\_\theta(w|D)$, which comprise the properties of Gaussian distributions $\mu \in \mathbb{R}\hat{}d$ and $\sigma \in \mathbb{R}\hat{}d$, denoted $N(\theta|\mu, \sigma^2)$, where d is the total number of parameters defining a probability distribution. The shape of these Gaussian variational posterior probability distributions, determined by their variance σ^2, expresses an uncertainty estimation of every model parameter.

Graphical Intuition of the above as defined by Graves (2011). Source

> *If you did not understand what exactly was said in the previous paragraph, don't worry. In the next post, we will cover all the basics needed to understand a Bayesian neural network.*

What to expect over the weeks

1. We will see how **Bayes by Backprop** can be efficiently applied to a CNN. We will introduce the **idea of applying two convolutional operations**, one for the mean and one for the variance.

2. We will see how the model learns **richer representations** and predictions from **multiple cheap model averaging**.

3. We will see that the proposed generic and reliable **variational inference** method for Bayesian CNN **can be applied to various CNN architectures** without any limitations on their performances. We will code the model in PyTorch and will compare the results with point estimate networks.

4. We will **estimate the aleatoric and epistemic uncertainties** in a Bayesian neural network. Furthermore, we will empirically show how uncertainty decreases, allowing the decisions made by the network to become more confident as the training accuracy increases.

5. We will learn that our method typically only **doubles the number of parameters** yet **trains an infinite ensemble using unbiased Monte Carlo estimates** of the gradients.

6. We will **apply L1 norm** to the trained model parameters and **prune the number of non-zero values.** Further, we fine-tune the model to reduce the number of model parameters without a reduction in the model prediction accuracy.

7. Finally, we will **apply the concept of Bayesian CNN** to tasks like **Image Super-Resolution** and **Generative Adversarial Networks** and we will compare the results with other prominent architectures in the respective domain.

Exciting! Right? Go to the **next post**.

. . .

> For all the impatient masters, check the entire theory here and implementation in PyTorch here.

kumar-shridhar/PyTorch-BayesianCNN

Bayesian Convolutional Neural Network with Variational Inference based on Bayes by Backpro...

github.com

Suggested Reads:

1. Weight Uncertainty in Neural Networks
2. Practical Variational Inference for Neural Networks
3. Bayesian Convolution Neural Networks using Variational Inference
4. Bayes by Backprop

Implementation:

PyTorch

. . .

Feel free to provide your invaluable comments and reach me here or on Twitter.

Machine Learning   Deep Learning   Neural Networks   Artificial Intelligence   AI