Reproduction of Bayes by Backprop paper

Matteo GHIA
Advanced Statistical Inference
EURECOM
Sophia-Antipolis, France
Matteo.Ghia@eurecom.fr

Abstract

This report aims to summarize the paper Weight Uncertainty in Neural Networks, which proposes a Bayesian approach to neural networks, specifically the Bayes by Backprop algorithm. We then reproduce the MNIST classification experiment using the library PyTorch.

1 Introduction

Neural networks have become a powerful machine learning tool, with strong results in many areas such as image classification and reinforcement learning. However, using deterministic weights makes the algorithms unable to model the uncertainty in the data, thus making overly confident predictions and leading to overfitting.

Several methods have been proposed to address this issue, such as earling stopping, dropout, and L2 regularization. While these techniques are useful, they do not provide a way to model uncertainty. Bayes by Backprop is instead a Bayesian inference approach that models the uncertainty in the weights of the neural network, making them a probability distribution over values rather than a single number. This leads to a model which can be considered as an ensemble of models, each of which has its weights sampled from the learnt distribution. This uncertainity in the weights can be used in reinforcement learning to introduce more variability in the decions made, leading naturally to exploration of the decision space.

2 Being Bayesian by Backpropagation

In Bayesian inference for neural networks, weights are a posterior distribution learnt during training. During the inference phase, the actual weights are sampled from this distribution. So each possible configuration of the weights is a possible model, thus taking expectations for making predictions is equivalent as using an infinitely large ensemble of models, each with its own weights sampled from the posterior distribution.

Variational learning is the process of finding the parameters θ of the weight distribution which minimize the Kullback-Leibler divergence with the true posterior distribution. This is done by maximizing the evidence lower bound (ELBO), which is a lower bound on the log-likelihood of the data given the model. The ELBO can be expressed as:

$$\mathcal{F}(D, \theta) = \mathrm{KL}[q(w|\theta)||P(w)] - \mathbb{E}_{q(w|\theta)}[\log P(D|w)]$$

The first term is the KL divergence between the approximate posterior distribution $q(w|\theta)$ and the prior distribution $P(w)$, while the second term is the expected log-likelihood of the data given the weights. The first one encourages the approximate posterior to be close to the prior, while the second one encourages the weights to be such that the data is well explained by the model.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

2.1 Unbiased Monte Carlo gradients

Minimizing exactly the ELBO is computationally unfeasible, thus gradient descent and various approximations are used. By using Monte Carlo sampling to evaluate the expectations and applying the re-parametrization trick for optimizing the ELBO, the algorithm Bayes by Backprop is obtained.

Previous work mentioned in the paper has used the re-parametrization trick to learn the parameters of stochastic hiddend units, but this paper extends the algorithm to the weights of the neural network, making this optimization problem several orders of magnitude larger. Moreover, the paper approximates the exact cost to

$$\mathcal{F}(D,\theta) \approx \sum_{i=1}^{n} \log q(w^{(i)}|\theta) - \log P(w^{(i)}) - \log P(D|w^{(i)})$$

All the terms depend on the Monte Carlo sampled weights $w^{(i)}$, which is a variance reduction technique called common random numbers.

2.2 Gaussian variational posterior

Since the cost function is approximated instead of being computed exactly starting from a Gaussian hypothesis, the prior and posterior distributions are not necessarily Gaussian.

The posterior distribution of the weights is modeled by a noise free diagonal Gaussian distribution with two degrees of freedom per weight. This only doubles the number of parameters while giving a good level of uncertainty to the weights. Using the re-parametrization trick, the weights are sampled and used to perform the forward pass of the neural network. The gradients of the cost function are then computed using the cost function described above, and the parameters of the posterior distribution are updated using gradient descent.

2.3 Scale mixture prior

The prior distribution is a fixed scale mixture of Gaussians with its hyperparameters chosen during tuning. The paper argues that optimizing the prior distribution during training leads to worse results because it learns to fit poor initial parameters quickly and falls into a local minima. Also in literature this is a debated topic, with criticism on its validity and usefulness.

The form of the prior distribution is as follows:

$$P(w) = \prod_j \pi N(w_j|0,\sigma_1^2) + (1-\pi)N(w_j|0,\sigma_2^2)$$

where π is the mixing parameter which controls the proportion of the two Gaussian distributions, w_j is the j-th weight of the neural network and σ_1 and σ_2 are the standard deviations of the two Gaussian distributions. By giving the first component a larger variance, the prior distribution resembles a spike-and-slab distribution.

2.4 Minibatches and KL re-weighting

The ELBO cost function can be used in minibatch optimization by re-weighting the KL divergence term. There are two proposal of weights, the first one being

$$\frac{1}{M}\mathrm{KL}[q(w|\theta)||P(w)]$$

where M is the number of minibatches; the second one is

$$\pi_i \mathrm{KL}[q(w|\theta)||P(w)]$$

where π_i is the weight of the i-th minibatch. This allows placing non uniform weights on the minibatches, and the paper proposes the following form of π_i:

$$\pi_i = \frac{2^{M-i}}{2^M - 1}$$

This leads to the KL having more importance in the first iterations thus giving more importance to the prior. In the subsequent minibatches progressively its weight is reduced, giving more importance to the log-likelihood term and subsequently to the data.

3 Contextual Bandits

Contextual bandits are a reinforcement learning problem where an agent is presented with a context and a set of choices which lead to different rewards. The agent must learn to choose the best action based on the context and the rewards received, but the context has to be independent of previous contexts, actions and rewards. It is also important to note that the agent does not have access the information on the rewards of the actions not taken, which is a problem known as "the absence of counterfactual".

During training, the agent build a model of the distribution of the rewards conditioned on actions and context, and this distribution can be modelled by a neural network. This network has to be trained not only to choose simply the best action based on the observations, but also to explore the action space otherwise the agent can under-explore and miss more rewarding actions.

One technique that allows to trade between exploration and best choices is Thompson sampling. At each step, new parameters are drawn and the action relative to those parameter is chosen. This is a kind of stochastic hypothesis testing and it usually requires the model parameters to be Bayesian. This process can easily be adapted to neural network and Bayes by Backprop, with the size of Monte Carlo sampling ruling the variance in gradient estimates and thus the trade-off between exploration and best choices. At the beginning of the training, the posterior is close to the prior, thus the actions are taken uniformily. As training progresses, the posterior distribution converges and the uncertainity on many parameters decreases, allowing more deterministic choices. In literature it is known that variatonal methods under-explore, but the paper argues that this didn't happen in their experiments.

4 Reproduction of the MNIST classification experiment

In this section I will describe the reproduction of the MNIST classification experiment proposed in the paper. The code is available on GitHub at `https://github.com/Maestro-Zacht/weight-uncertainty-nns/blob/main/experiment.ipynb`.

4.1 Dataset and data preprocessing

The MNIST dataset is a well known dataset of handwritten digits, with 60,000 training images and 10,000 test images. Each image is a 28x28 grayscale image, and the task is to classify the digit in the image from 0 to 9 included.

The images are preprocessed by flattening them and dividing the pixel values by 126 as described in the paper. Subsequently, the training dataset is split into 50,000 training images and 10,000 validation images.

4.2 Model

The model consists of two Bayesian layers with ReLU activation function, each with 1200 hidden units, and a third layer with 10 softmax output units. The implementation of the model follows as much as possible the DRY principle, with the model, layers, prior and posterior distributions being defined in separate classes and their behaviour implemented in parametrized methods.

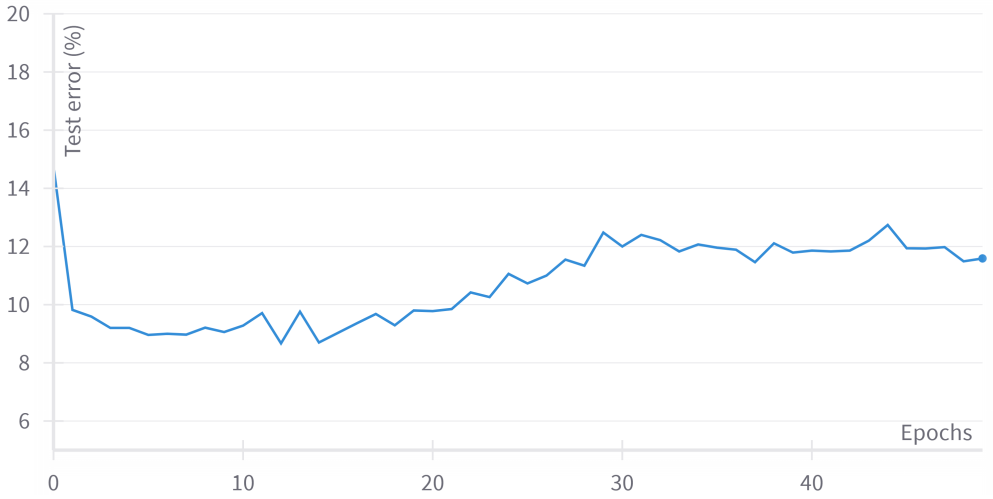

Figure 1: Test error on the MNIST dataset.

4.3 Training

Minibatch stochastic gradient descent is used to train the model by using the Adam optimizer, with batch size 128. The log-likelihood cost is the cross-entropy between the predicted distribution and the true distribution of the labels, as described in the paper. The weight of the KL divergence term is set to the first option, which is $\frac{1}{M}$, where M is the number of minibatches.

Hyperparameter tuning is done using a grid search on the parameters listed in the paper, with the following values:

- Learning rate: 10^{-3}, 10^{-4}, 10^{-5}
- Prior mixing parameter π: $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$
- Prior 1 negative log standard deviation $-\log \sigma_1$: 0, 1, 2
- Prior 2 negative log standard deviation $-\log \sigma_2$: 6, 7, 8
- Number of Monte Carlo samples: 1, 2, 3, 5

The values are the very same as in the paper, except for the number of Monte Carlo samples. I found 10 to always perform worse, so I changed the 10 to a 3 to allow finer selection in lower values.

The best hyperparameters found are:

- Learning rate: 10^{-3}
- Prior mixing parameter π: $\frac{1}{2}$
- Prior 1 negative log standard deviation $-\log \sigma_1$: 0
- Prior 2 negative log standard deviation $-\log \sigma_2$: 6
- Number of Monte Carlo samples: 3

The model is then trained for 50 epochs on the whole training dataset of 60,000 images, and the test error on every epoch is represented in Figure 1.

4.4 Discussion and problems encountered

The results are similar in shape to the ones presented in the paper, but they differ of 1 order of magnitude in the values. Training for more epochs did not lead to better results, and the model was clearing overfitting and reaching up to 40% test error rate. This is likely due to implementation details and tricks that the authors used to improve the performance.

Another aspect that was not mentioned in the paper is the weight initialization. The authors do not specify how the weights are initialized and some of them lead to very poor results, even random guessing (90% test error rate). After some trial and error, I found that initializing the weights with a normal distribution with arbitrary mean and standard deviation proportional to the size of the layer was a good choice.

Additionally, the two different ways to compute the KL divergence weight in the cost function are not interchangeable. While the first one leads to a good performance most of the times, the second one sometimes even leads to exploding gradients and model parameters going to NaN. Due to this behaviour, I decided to use the first one only.

Similarly, also the optimizer is not well specified in the paper. It is mentioned that the SGD optimizer is used, but not which of its variants. Using PyTorch's plain SGD optimizer leads to weights going to NaN most of the time, while using Adam is more reliable.