Project 2: Bayesian Neural Networks

Mihail Feraru

January 2022

1 Introduction

In this work we compared classical neural networks (NNs) trained using stochastic gradient descent (SGD) methods and bayesian neural networks (BNNs) *trained* [1] with probabilistic Monte-Carlo Markov-Chain (MCMC) methods. We aim to analyse the probabilistic approach in the following scenarios:

1. given an existing NN, train a BNN of the same architecture using data generated by the NN;
2. perform a sanity check on a simple binary classification and a multi-label classification problem such as XOR classification and Iris dataset;
3. a harder binary classification problem such as the two moons with enough noise;
4. a harder multi-class classification problem.

We will compare the results to the classic NN approach, considering training time, train/validation/test accuracies, overfit resistance, prediction variance and uncertainity estimation.

2 Technical framework

To ease running experiments, we created a small framework (entirely) in Julia which allows quick setup and execution of comparison experiments for NNs and BNNs. On top of Flux library [2] and MLJ.jl [3] we created utility functions for loading some well-known toy datasets (Iris, Two Moons etc), data scaling, NN training, evaluation and reporting.

For BNNs, we built our framework on top of Gen [4]. We created a generic generative function for sampling the parameters and outputs of any BNN implemented in Flux. The generative function is aimed at any network that solves an N-label classification task, and if we let $f : \mathbf{R}^n \times \mathbf{R}^k \to \mathbf{R}^m$ with $f(X, W) = O$ denote a neural network with weights and biases W (of size k when all are considered as a single vector), inputs X and outputs O, we describe the generative model as:

$$W \sim \text{Normal}(\mu, \sigma^2)$$
$$y \sim \text{Categorical}(\text{softmax}(f(X, W)))$$

The generative function is used as the prior distribution of the BNN and alongside Metropolis-Hastings algorithm it is used to infer the posterior $p(W|\hat{y}, X)$. To have a fair comparison with gradient descent methods, Metropolis-Hastings was implemented in a mini-batch mode, thus at each iteration of the algorithm, we take a random subsample of X and its corresponding labels y and optimize the likelihood for the subsample instead of the entire training data. This approach is guaranteed to also converge as stated by Zhang et al. [1].

The framework's core can be found in the attached source code in the directory *BayesianNetworks/src/training.jl*. Example implementations for the datasets described in this work can be found in *BayesianNetworks/src/examples* and *BayesianNetworks/test*.

[1] Actually, their weights and biases are inferred, so training is not quite the right term here.
[2] Flux can be found at the following address: `https://fluxml.ai/Flux.jl/stable/`.
[3] MLJ.jl can be found at the following address: `https://alan-turing-institute.github.io/MLJ.jl/dev/`.
[4] Gen framework can be found at the following address: `https://www.gen.dev/`.

3 Experiments

In the Table 1 below you can see the results of a training session with both methods (NNs and BNNs) for datasets of 5000 examples, with a train/validation/test split with the ratios 8/1/1. The training was performed for the same number of epochs for both approaches and with a constant batch size of 256.

Algorithm	Problem	Epochs	Time	Train accuracy	Validation accuracy	Test accuracy
NN + GD	XOR	500	4s	97.54%	97.40%	97.80%
NN + GD	Two Moons	100	<0s	98.62%	99.00%	99.00%
NN + GD	10 Blobs	100	<0s	100.00%	99.00%	98.00%
BNN + MH	XOR	500	10s	91.94%	92.60%	93.40%
BNN + MH	Two Moons	100	6s	98.37%	99.00%	98.00%
BNN + MH	10 Blobs	100	10s	94.12%	89.00%	87.00%

Table 1: Comparison of NNs and BNNs on three classification problems. Simple NNs trained using Gradient Descent seem to over-perform BNNs trained using Metropolis-Hastings and be faster.

If we look at Figure 1, we see that MH is unable to brind the parameters close the optimal values. On the other hand, if we look at Figures 3 and 2 we see that is performs slighly better on a problem where noise is involved and it initially achieves a better loss than GD faster.

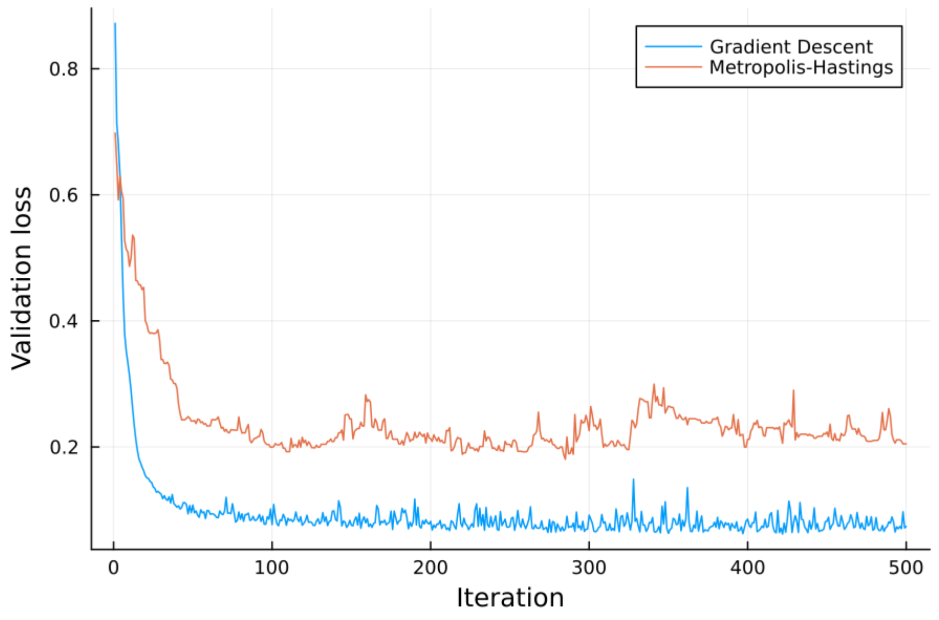

Figure 1: Evolution of validation loss for NNs and BNNs on the XOR problem.

4 Conclusions

We can summarize the experiments above as follows:

1. GD achieves usually better accuracies than MH;
2. MH is unable to overfit the training set, this could be due to numerical instability of the algorithm;
3. MH could be replaced by HMC or Variational Inference to achieve better results;
4. MH needs to see fewer examples to learn, but it is stuck in a much worse local minimum (it should converge to the global minimum but it could take ages);
5. MH provides a sample of models which reduce variances of the predictions.

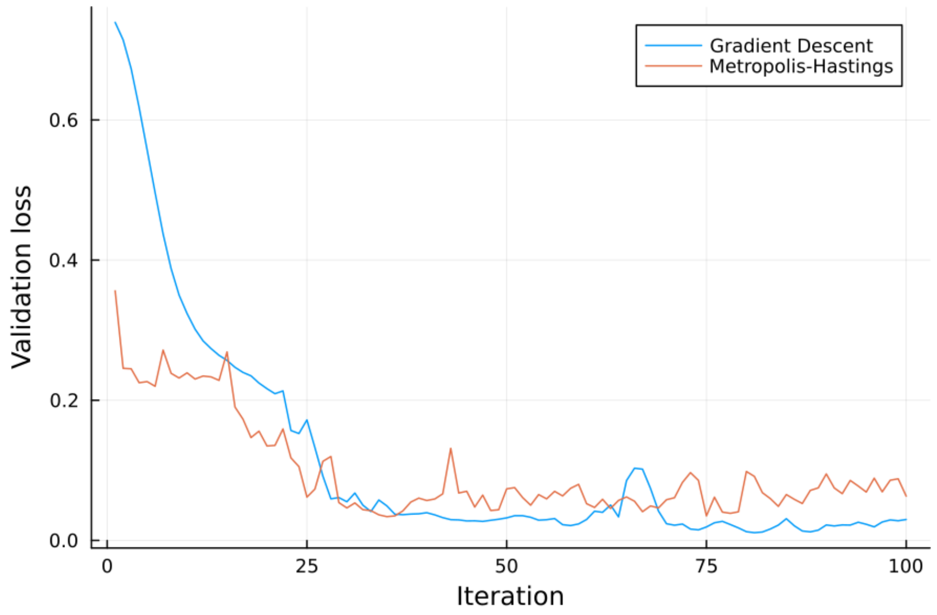

Figure 2: Evolution of validation loss for NNs and BNNs on the Two Moons problem.

References

[1] Ruqi Zhang, A. Feder Cooper, and Christopher De Sa. "Asymptotically Optimal Exact Minibatch Metropolis-Hastings". In: (2020). DOI: 10.48550/ARXIV.2006.11677. URL: https://arxiv.org/abs/2006.11677.

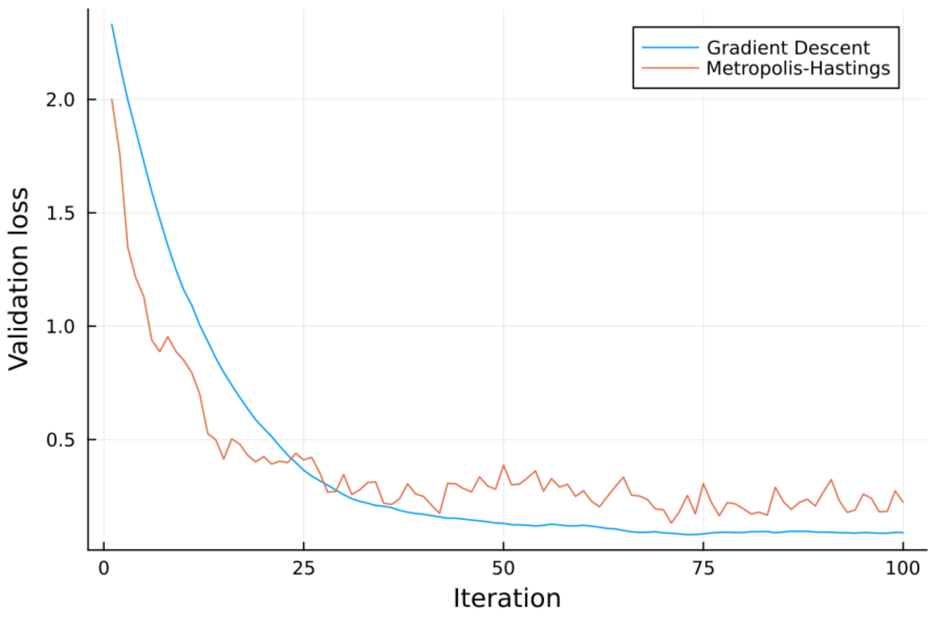

Figure 3: Evolution of validation loss for NNs and BNNs on the 10-Blobs problem.