Improving the Identifiability of Neural Networks for Bayesian Inference

Arya A. Pourzanjani[*], **Richard M. Jiang**[*], **Linda R. Petzold**
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
arya@ucsb.edu,rmjiang@ucsb.edu,petzold@cs.ucsb.edu

1 Introduction

Accurate inference of the parameters in the highly complex and multi-modal likelihoods of Neural Networks(NNs) is incredibly difficult for any algorithm. In part, this challenge is caused by the significant over-parameterization of the model, resulting in many equivalent solutions and thus a model unidentifiability problem. In this paper, we explore the unidentifiability problem for NNs as it manifests in two ways: arbitrary permutations of the hidden nodes, which we denote as weight-space symmetry, and arbitrary scaling under rectified linear-unit (ReLU) nonlinearites, which we denote as scaling symmetry. We show how these unidentifiabilities pose issues for both Markov Chain Monte Carlo (MCMC) and Variational Inference (VI). Finally, we introduce two reparameterizations of the model in the form of parameter constraints and prove that they resolve the aforementioned unidentifiability issues, showing some experiments and offering implementations in the form of coordinate transforms.

2 Unidentifiability of Neural Networks

Let (x_1, x_2) be the input nodes to a NN with 1 hidden layer of 2 nodes, (h_1, h_2) and a single output node y. We define the following standard setup for a single layer.

$$\begin{aligned} h_1 &= \sigma(W_{11}x_1 + W_{21}x_2 + b_1) \\ h_2 &= \sigma(W_{12}x_1 + W_{22}x_2 + b_2) \\ y &= w_1h_1 + w_2h_2 + c = w_2h_2 + w_1h_1 + c \end{aligned}$$

Under this setup, we describe and define the weight-space symmetry and the scaling symmetry unidentifiabilities and note that these holds in wider and deeper structures as well. We note that it is likely that further unidentifiabilies exist in the formulation of the neural networks. In this work, we only tackle these two with hopes of further investigation in the future.

2.1 Weight-Space Symmetry

Definition 2.1. Weight-space symmetry is the unidentifiability induced in NNs by swapping both the input and output weights and biases of any two hidden units in the same layer [2, p.277]. In our setting, this is equivalent to switching the labels of h_1 and h_2, which leaves y invariant.

Weight-space symmetry arises due to the nature of the output operations in neural networks. Hidden nodes are fully exchangeable as they are added together to obtain the output of the next layer. As this is a label switching problem, this gives way to a factorially growing number of equivalent modes as the number of hidden nodes and hidden layers increase.

[*]Equal Contributors

Second workshop on Bayesian Deep Learning (NIPS 2017), Long Beach, CA, USA.

2.2 Scaling Symmetry

Definition 2.2. Scaling symmetry is the unidentifiability induced in ReLU NNs by arbitrary scaling of the input weights and biases of a hidden node by a real scalar α and scaling the output by $1/\alpha$. This can be seen in our example by multiplying W_{11} and W_{21} by α and multiplying w_1 by $1/\alpha$. Under the ReLU, this scaling leaves y invariant.

More generally, scaling symmetry is exclusive to nonlinearities with the property $\sigma(\alpha x) = \alpha\sigma(x)$ where α is a real scalar. Notably, this includes the ReLU. In this case, we can arbitrarily scale all of the weights and biases going into a hidden layer by a real scalar, α, and then scale the output weights by $1/\alpha$, resulting in the exact same output. Geometrically, this produces a high-dimensional hyperbola of equivalent solutions [2, p.277].

3 Adding Parameter Constraints for Identifiability

To resolve these two issues, we propose a set of reparameterizations of NNs in the form of parameter constraints on the weights and biases. We implement these constraints in the form of a coordinate transformation from constrained to unconstrained space. This is a common technique in the Bayesian inference literature that allows for inference methods to remain unaltered [1; 4]. Such transforms require both an invertible mapping and a corresponding change of probability measure, which fortunately in our case has been solved for these two cases.

3.1 Bias Ordering Constraint

To deal with the arbitrary permutations from weight space symmetry, we introduce the constraint that biases must be ordered. More specifically, the **bias ordering constraint** restricts the bias vectors in each hidden layer to be ordered in such a way that

$$b_L^{(1)} < b_L^{(2)} < \cdots < b_L^{(N_L)}. \tag{1}$$

Theorem 3.1. *Placing an ordering constraint on the biases of a neural network layer eliminates the weight space symmetry.*

The bias ordering constraint is implemented by performing inference on the log of the differences of entries and transforming to the constrained parameters for evaluation [1].

3.2 Unit-Length Weight Vector Constraint

To resolve the scaling symmetry, we introduce the **unit-vector weight constraint**, which constrains the vector of weights of going into a hidden node to be of unit norm.

Theorem 3.2. *Any configuration of weights for a deep ReLU neural network with non-zero weight vectors has an equivalent unit-vector constrained weight configuration with the last layer of weights unconstrained.*

The unit-vector constraint is implemented by performing inference on an unconstrained space via the Givens' Transform, a transform from the space of unit-vectors and orthonormal matrices to unconstrained space [5].

4 Experiments

4.1 Synthetic Data

We generate 100 data points from a fully connected single ReLU layer Neural Network with 2 hidden nodes using the following parameters: $W_1 = [[1/\sqrt{2}, -1/\sqrt{2}], [1/\sqrt{2}, 1/\sqrt{2}]]$, $b_1 = [0, 1]$, $W_2 = [[1/\sqrt{2}, 1/\sqrt{2}]$, $b_2 = [5]$ with added univariate Gaussian noise. We then infer the constrained and unconstrained models, placing $\mathcal{N}(0, 1)$ priors on the parameters, using HMC, Mean-Field ADVI, and Full-Rank ADVI in Stan [1]. In the unconstrained model, several equivalent solutions with equal posterior density exist along a high-dimensional hyperbola. This results in a posterior with high

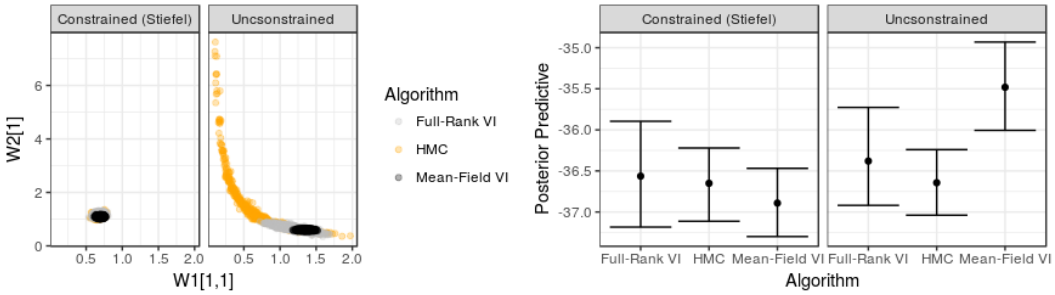

Figure 1: (Left) Posterior samples of $W_1[1,1]$ and $W_2[1]$ illustrate how the model identifiability issue results in a pathological posterior over many equivalent solutions and with unnaturally high-curvature. This posterior is not only difficult to sample from efficiently, but difficult for VI to approximate, which results in biased posterior predictive distributions of new samples (Right). Using unit vectors via the Givens Transform results in a much more well-behaved posterior that is easier to sample from and approximate using VI. This amounts to more accurate estimates and posterior predictive uncertainties of new data.

Split	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Unconstrained	0.37	0.31	0.30	0.39	0.34	0.42	0.43	0.34	0.46	0.33	0.38	0.47	0.42	0.37	0.36
Constrained	**0.36**	**0.28**	**0.28**	**0.31**	**0.29**	**0.31**	**0.39**	**0.31**	**0.45**	**0.29**	**0.38**	**0.31**	**0.37**	**0.27**	**0.27**

Table 1: Mean-Square Error (MSE) on 15 test sets using unconstrained and constrained model.

curvature that is difficult to explore or approximate using VI, which itself results in biased posterior predictive distributions of target variables (Figure 4.1 [right]). The constrained approach using the Givens Transform removes many of the equivalent solutions, resulting in a much more well-behaved posterior. In practice, we found the number of effective samples and average NUTS tree depth to be lower in the constrained case.

4.2 Boston Housing

The Boston Housing dataset consists of the price and other features of 506 unique homes in the Boston area [3]. We modeled this task with a 50 node single hidden layer ReLU neural network and inferred the parameters using both the unconstrained and constrained parameterizations under 15 different 80/20 train/test splits. Inference was done using the NUTS algorithm in Stan. We use the mean of the posterior predictive distributions and compared them to the true values from the test set to obtain a Mean-Square Error (MSE) in both settings. In every case the constrained version using the Givens Transform outperformed the unconstrained version (Table 4.2). Similarly we use the posterior predictive distributions on the test sets to compare how often the 95% credible interval of the posterior covered the true value. The constrained distribution has test examples outside of the 95% credible interval far less often, as the constrained posterior is much easier to explore (Table 4.2). Paired Wilcoxon tests of the MSE and coverage using the unconstrained versus the constrained models results in p-values of 0.009 and 0.003, suggesting that better performance of the constrained model was not due to pure chance alone.

5 Discussion

Reparameterization of problems in Bayesian Inference is pivotal to obtaining good performance and results for inference. In this short work, we explore two ways in which we can reparameterize Neural Networks to eliminate excess multi-modalities and equivalences. We note that, although the reparameterization eliminates some of the clear multi-modalities, they do not make the likelihood nearly convex nor simple. Multiple modes still exist, and chains as well as approximate posteriors will still result in mode finding behavior. We hope that there is more research into capturing the true posteriors, as this epistemic uncertainty can manifest itself in the predictions made by Neural Networks under poor inference methods.

Split	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Unconstrained	**0.05**	0.08	0.11	0.09	0.13	0.13	0.12	**0.09**	0.15	0.06	0.13	0.11	0.13	0.10	0.13
Constrained	0.08	**0.05**	**0.05**	**0.06**	**0.08**	**0.10**	**0.04**	0.11	**0.07**	**0.06**	**0.09**	**0.03**	**0.12**	**0.05**	**0.09**

Table 2: Posterior predictive coverage on 15 test sets using unconstrained and constrained model.

References

[1] Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M. A., Guo, J., Li, P. and Riddell, A. [2016], 'Stan: A probabilistic programming language', *Journal of Statistical Software* **20**, 1–37.

[2] Goodfellow, I., Bengio, Y. and Courville, A. [2016], *Deep learning*, MIT press.

[3] Harrison, D. and Rubinfeld, D. L. [1978], 'Hedonic housing prices and the demand for clean air', *Journal of environmental economics and management* **5**(1), 81–102.

[4] Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A. and Blei, D. M. [2016], 'Automatic differentiation variational inference', *arXiv preprint arXiv:1603.00788* .

[5] Pourzanjani, A. A., Jiang, R. M., Atzberger, P. J. and Petzold, L. R. [2017], 'General Bayesian Inference over the Stiefel Manifold via the Givens Transform', *ArXiv e-prints* .

A Proof to Theorem 3.1

Proof. Suppose we impose a bias ordering constraint on a neural network. Let h_i and h_j be two hidden nodes such that $h_i = \sigma(W_i X + b_i)$, $h_j = \sigma(W_j X + b_j)$, and $y = \sigma(\sum_k w_k h_k)$ with $b_i < b_j, i < j$. Suppose we permute the input and output weights and biases of h_i and h_j, swapping the labels but producing an equivalent solution from weight-space symmetry. However, this violates the bias-ordering constraint because now $b_i > b_j, i < j$. Thus permutations of arbitrary hidden nodes are not a valid solution. □

B Proof to Theorem 3.2

Proof. We proceed with a constructive argument. Let W_1 be the $N \times M$ weight matrix of the first hidden layer in the neural network with columns $w_1^{(i)}$. We transform W_1 to W_1', by dividing each column by it's norm so that $w_1^{(i)} = w_1^{(i)}/\|w_1^{(i)}\|$. Under this transform, $h_1^{(i)'} = \sigma(X w_1^{(i)'} + b_1^{(i)}/\|w_1^{(i)}\|) = (1/\|w_1^{(i)}\|)h_1^{(i)}$ by the property of the ReLU. Let W_2 be the second weight matrix. Then for the second layer with respect to W_1', $h_2^{(i)'} = \sigma(\sum_k \|w_1^{(k)}\| h_1^{(k)'} w_2^{(ik)} + b_2^{(i)})$, which tells us that this is equivalent to multiplying the i-th row of W_2 by $\|w_1^{(i)}\|$. Denote $\hat{W}_2$ to be W_2 with rows scaled by the previous layer's norms. At this point, we can consider $\hat{W}_2$ as the first layer of a NN, as no transformation will propagate back to previous layers due to the structure of the NN. Thus, we can repeat the same process and transform $\hat{W}_2$ to $\hat{W}_2{}'$. We continue this process until the final layer, noting that each previous layer now has unit-length columns. Let $\hat{W}_L$ be the weight matrix of the final layer as a process of transforming all layers before it. The output can now be written as $y = \sum_i h_L \hat{w_L}^{(i)} + b_L$. We keep this layer unconstrained as y is fixed. The entire network now has unit-vector columns except the final set of weights, which absorbs the scaling of the previous layers, but produces an equivalent for output. □