Bayesian Neural Networks using HackPPL with Application to User Location State Prediction

Beliz Gokkaya[⋆1], **Jessica Ai**[⋆1], **Michael Tingley**[1], **Yonglong Zhang**[2]

Ning Dong[1], **Thomas Jiang**[1], **Anitha Kubendran**[1], **Arun Kumar**[1]
[1]Facebook, [2]University of Southern California
[⋆]{belizg, jaix}@fb.com

1 Introduction & Related Work

At Facebook, we are becoming increasingly interested in incorporating uncertainty into models used for decision making. As a result, we are building an in-house universal probabilistic programming language that aims to make modeling more accessible to developers and to unify the tooling experience for existing users of Bayesian modeling. In addition, we form interfaces for common probabilistic models on top of this language and apply these to datasets within the company.

For the less familiar reader, probabilistic programming languages (PPLs) provide a convenient syntax for allowing users to describe generative processes composed of various sources of uncertainty. The user may then pose probabilistic queries about their world that will be resolved using an inference engine. Some of the more mature languages include domain specific languages such as WinBUGS [1], JAGS [2] and Stan [3], which place some restrictions on the models a user may write in order for inference to run more efficiently. On the other hand, the newer universal PPLs such as Church [4], WebPPL [5] and Anglican [6] extend existing general-purpose languages and resolve queries through a generic inference engine. In doing so, users are constrained only by the limitations of the underlying language, although this may not always result in the most efficient model and this tradeoff between model expressivity and inference efficiency is an ongoing area of research.

Traditionally, Bayesian neural networks (BNNs) are neural networks with priors on their weights and biases [7, 8]. Their main advantages include providing uncertainty of predictions rather than point estimates, built-in regularization through priors, and better performance in problem settings such as the robot arm problem [7, 9], but are generally expensive in terms of compute time. While probabilistic interpretations of neural networks have been studied in the past, BNNs have seen a resurgence in popularity in recent years, particularly with alternative probabilistic approaches to dropout [10] and backpropagation [11], and the renewed investigation of variational approximations [12] which have made computation more tractable. There have also been more recent advances in combining probabilistic programming and deep learning, notably by Edward [13] and Pyro [14]. These languages are built on top of existing tensor libraries and have so far focused on variational approaches for scalable inference.

In this study, we present HackPPL as a probabilistic programming language in Facebook's server-side language, Hack. One of the aims of our language is to support deep probabilistic modeling by providing a flexible interface for composing deep neural networks with encoded uncertainty and a rich inference engine. We demonstrate the Bayesian neural network interface in HackPPL and present initial results of a multi-class classification problem to predict user location states using Markov Chain Monte Carlo. Through HackPPL we aim to provide tools for interacting and debugging Bayesian models and integrate them into the Facebook ecosystem.

Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montréal, Canada.

2 HackPPL

2.1 Language

HackPPL is a universal probabilistic programming language written in Facebook's server-side language, Hack [15]. We have extended the Hack language by embedding random events into the language in order to provide expressivity and flexibility for statistical modeling. More specifically, our work is based on the continuation passing style transformation [16] and we use coroutines backed by multi-shot continuations, which we have added to the Hack language. Indeed, these coroutines constitute the building blocks of HackPPL's trace-based implementation.

As a universal PPL, we choose to separate the modeling and inference in order to abstract away the details of inference from users. On the modeling side, we adopt a generative modeling approach that allows for stochastic control flow. Two important language constructs for forming a model in HackPPL are 'sample' and 'observe', in which 'sample' statements enable repeated sampling from probability distributions, whereas 'observe' statements enable conditioning on data. By implementing these language constructs as coroutines, we can discriminatively explore different parts of the posterior. Below we provide an example program in HackPPL, which demonstrates Bayesian linear regression.

```
$m = sample(new Normal(0.0, 2.0), 'm');
$b = sample(new Normal(0.0, 2.0), 'b');
$sigma = sample(new Gamma(1.0, 1.0), 'sigma');
$mu = $x->scalarMult($m)->scalarAdd($b);
observe(new TensorNormal($mu, Tensor::ones_like($mu)->scalarMult($sigma)), 'y', $y);
```

Figure 1: Linear regression model in HackPPL

We also aim to provide a generic Bayesian inference engine and currently support Markov Chain Monte Carlo (MCMC) methods, including Hamiltonian Monte Carlo, Sequential Monte Carlo, and Variational Inference. The tensor and automatic differentiation implementations in HackPPL are backed by PyTorch [17] to take advantage of PyTorch's support for vectorization, GPUs and distributed computing. In addition, we provide tools for quantitative and visual inference introspection and for performing posterior predictive checks. We provide an example sampler in HackPPL below for obtaining the posterior distribution of the coefficients of the linear regression model.

```
$model = new LinearRegression($x, $y);
$history = PPLInfer::hmc($model)->history()->run($num_iterations);
$history->getSample('m');
```

Figure 2: Sampler for linear regression model in HackPPL

2.2 Bayesian Neural Networks

HackPPL's BNN implementation provides abstractions for various layer types such as dense and convolutional layers, which allows composition of different architectures. Within each layer, users can make use of the API to specify priors, and declare deterministic and stochastic hyperparameters. We further build on top of these abstractions to provide a convenient syntax for creating and configuring feedforward, recurrent and convolutional neural networks.

We demonstrate the BNN interface in HackPPL for a feedforward regressor with architecture shown in Figure 3 and the corresponding code provided in Figure 4. Here, the weights and biases are defined to have hierarchical Gaussian priors, where hyperpriors control the mean and variance of the priors. Automatic relevance determination [7] is used to understand the relative relevancy of input features. The observed data is defined to follow a Gaussian distribution centered around the predicted value.

3 Application to User Location State Prediction

We present our initial attempts to apply the BNN classifier to a user location prediction problem at Facebook where the motivation is to learn contextual information around a user's location at a particular point in time. For testing purposes, our dataset contains 88 features describing the context

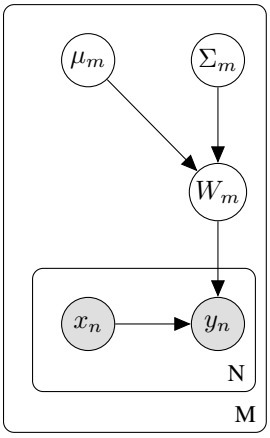

Figure 3: Graphical model of the Bayesian neural network

```
$likelihood_dist = (
  ($y_predicted, $param)
  ==> new TensorNormal($y_predicted), $param['sigma']
);
$model = new BayesianNN(
  new FeedForwardNetwork(
  $x,
  $nodes_per_layer,
  $likelihood_dist,
  $likelihood_param,
  $use_ARD,
));
$model->setPriors(
  ($params)
    ==> new TensorNormal($params['mu'], $params['sigma']),
  dict['mu' => new Normal(0., 1.),
        'std' => new Gamma(1., 1.)],
  $layers_to_change,
);
```

Figure 4: Bayesian neural network for regression in HackPPL

of a particular user and we classify each user's location into one of 5 states. We test a neural network architecture consisting of three layers having 32 nodes each and with tanh activations. Weights and biases are modeled using hierarchical Gaussian priors whose variances are scaled using hyperpriors. We then use MCMC inference to obtain the posterior distribution of the neural network parameters and inspect the posterior predictive distributions of the training and evaluation datasets.

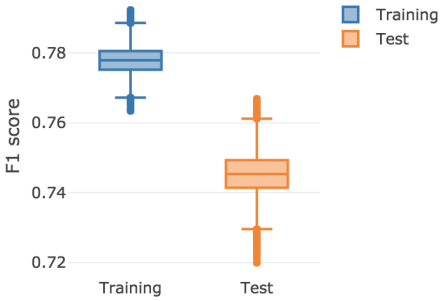

Figure 5: Boxplot of F1-scores of posterior predictive samples against training and test data

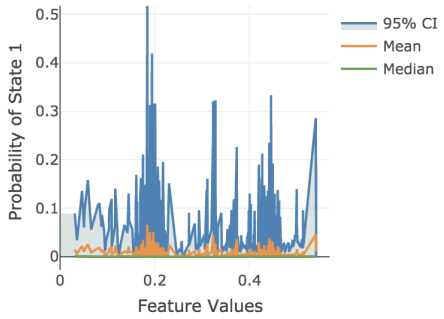

Figure 6: Prediction uncertainty with respect to an important feature

To assess model fit, we obtain the posterior predictive distribution and calculate the distribution of F1-scores across the replicated dataset. We present the distribution of F1-scores obtained across training and test datasets Figure 5. The F1-scores are comparable to results from existing frequentist models and, as expected, we observe a larger 95% credible interval for the test data. In Figure 6, we dive deeper by plotting the prediction of a particular state along with its uncertainty with respect to one of the important input features. The predictions yield significant uncertainty for certain values of this feature. Overall, we are able to understand the uncertainty in the predictions, which facilitates diagnosing and understanding the limitations of a statistical model.

4 Conclusion

This study introduces HackPPL as a universal probabilistic programming language and demonstrates the Bayesian neural network API built using HackPPL. We also present our initial efforts to model user location states using a Bayesian neural network classifier and show that by preserving the uncertainty in the estimates, we are able to understand the limitations of the model as well as represent our confidence in the predictions. This is vital to us at Facebook, as we are particularly interested in understanding where our predictive models behave poorly and in representing our confidence in predictions. Through adopting a Bayesian approach, HackPPL supports these motivations by integrating tools targeting model improvement and debugging for maintainability of statistical models into a machine learning developer's workflow.

References

[1] D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter, "Winbugs-a bayesian modelling framework: concepts, structure, and extensibility," *Statistics and computing*, vol. 10, no. 4, pp. 325–337, 2000.

[2] M. Plummer, *JAGS Version 4.3.1 user manual*, http://people.stat.sc.edu/hansont/stat740/jags_user_manual.pdf.

[3] Stan Development Team, *Stan Modeling Language Users Guide and Reference Manual*, http://mc-stan.org/.

[4] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," *arXiv preprint arXiv:1206.3255*, 2012.

[5] N. D. Goodman and A. Stuhlmüller, "The design and implementation of probabilistic programming languages," 2014.

[6] D. Tolpin, J.-W. van de Meent, H. Yang, and F. Wood, "Design and implementation of probabilistic programming language anglican," in *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages*. ACM, 2016, p. 6.

[7] R. M. Neal, "Bayesian learning for neural networks," Ph.D. dissertation, University of Toronto, 1996.

[8] C. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[9] D. J. C. Mackay, "Bayesian methods for adaptive methods," Ph.D. dissertation, California Institute of Technology, 2012.

[10] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *International conference on machine learning*, 2016, pp. 1050–1059.

[11] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *International conference on machine learning*, 2015.

[12] A. Graves, "Practical variational inference for neural networks," in *Advances in neural information processing systems*, 2011.

[13] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei, "Deep probabilistic programming," *arXiv preprint arXiv:1701.03757*, 2017.

[14] Uber AI Labs, *Pyro*, http://pyro.ai/, 2018.

[15] Facebook Hack Language Team, *Hack*, https://hacklang.org/, 2018.

[16] D. Wingate, A. Stuhlmüller, and N. Goodman, "Lightweight implementations of probabilistic programming languages via transformational compilation," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 770–778.

[17] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration," 2017.