MXFusion: A Modular Deep Probabilistic Programming Library

Zhenwen Dai
Amazon
zhenwend@amazon.com

Eric Meissner
Amazon
erimeiss@amazon.com

Neil D. Lawrence
Amazon
lawrennd@amazon.com

1 Introduction

Deep neural networks (DNNs) have seen great successful in various areas such as computer vision, speech recognition, and artificial intelligence. A major driver of such success is the rise of deep learning libraries, starting from Theano (Theano Development Team, 2016) and following by many popular libraries such as TensorFlow (Abadi et al., 2015), PyTorch (Paszke et al., 2017). The unique benefits that these libraries bring are: (1) completely transparent training and prediction, (2) modular DNN construction with interchangeable components. A DNN component can vary dramatically in terms of size and complexity, ranging from a single matrix operator, to a highly complex deep neural network such as convolutional neural networks (CNN) or long short-term memory (LSTM). All the components share the same interface, which enables a user to replace one component by another with almost no changes on other parts of the code. Modularity in the programming languages used by deep learning libraries revolutionizes the development, communication and deployment of DNNs. The ability to package a state-of-the-art implementation into a standard module and distribute enables rapid development with much lower chance of having bugs and performance issues comparing with re-implementing from scratch, which makes research results more reproducible. Communication about ongoing research method and results becomes much easier, as running a different experiment or modifying a network only requires to change a few lines of code. The standardized interface of components enables better testing and maintainability. It eases the transition of a research implementation into an industrial software.

Probabilistic programming language (PPL, Lunn et al., 2000; Pfeffer, 2001; Goodman et al., 2008; Pfeffer, 2009; Murray, 2013; Paige & Wood, 2014; Minka et al., 2014; Carpenter et al., 2017; Ge et al., 2018) is a similar domain specific programming language that aims at describing probabilistic models and automating inference in those models. Despite the difference of DNNs and probabilistic models in their mathematical formulations[1], a similar modularity structure exist in both of them. PPLs exploit such modularity structure in probabilistic models by providing various probabilistic distributions as building blocks. However, unlike DNN, where a gradient-based training algorithm is straight-forward after the network is defined, exact inference in a probabilistic model is usually intractable. As a result, in the literature of machine learning, a probabilistic model is often presented with a dedicated inference method that maximizes accuracy and efficiency of inference by exploiting the specific mathematical formulation of the model. For instance, Dirichlet process is firstly introduced with a Gibbs sampling algorithm by MacEachern (1994); Ishwaran & James (2001) and lately with a variational inference method by Blei & Jordan (2006). Unfortunately, current PPLs are not able to take advantages of those kind of inference methods, because those inference methods are tied to the specific probabilistic models, while the merit of PPL is its ability to program flexible probabilistic models. Instead, current PPLs rely on generic inference methods that do not need to know the specific form of a probabilistic model ahead of time, in the form of variational inference such as automatic differentiation variational inference (Kucukelbir et al., 2017) or Markov Chain Monte Carlo (MCMC) sampling. Although generic inference methods are able to handle a wide family of probabilistic models, they often run slower or produce less accurate results comparing to a dedicated inference method on the same model. This performance gap is a major obstacle for adoption of PPLs.

[1] A DNN represents a deterministic function, while a probabilistic model describes a probabilistic distribution.

Preprint. Work in progress.

In this paper, we propose to close this performance gap by incorporating dedicated inference methods into PPLs. We exploit modularity not only in model definition but also in inference by including a new type of re-usable building blocks for probabilistic models called *probabilistic modules*. A probabilistic module consists of a set of random variables with associated probabilistic distributions and dedicated inference methods designed specifically for efficiency and accuracy on that set of random variables. This allows us to package a sophisticated probabilistic model with their proposed inference methods as a re-usable building block, e.g., Dirichlet process with variational inference (Blei & Jordan, 2006). We demonstrate this idea by presenting a modular probabilistic programming language MXFusion[2], in which we implement probabilistic modules under the framework of variational inference. In MXFusion, one could seamlessly combine generic variational inference methods with dedicated variational inference methods, which bridges the performance gap. This will help to bring the a lot of the benefits of modularities that we enjoy with deep learning libraries to probabilistic programming and improve the usabilities of PPLs on real world problems.

2 Probabilistic Module with Variational Inference

The concept of hiding the details of inference of a probabilistic module by specialized inference methods is nice. The challenge is that not all the approximate inference method are compatible with each other. In this section, we present an approach to implement probabilistic module with the framework of variational inference. The main idea of variational inference is to approximate an intractable posterior distribution of latent variables with a parametric approximation, referred to as a variational posterior distribution. VI is often framed as a lower bound of the logarithm of the marginal distribution, e.g,

$$\log p(y|z) = \log \int_x p(y|x)p(x|z) \geq \int_x q(x|y,z) \log \frac{p(y|x)p(x|z)}{q(x|y,z)} = \mathcal{L}(y,z), \tag{1}$$

where $p(y|x)p(x|z)$ forms a probabilistic model with x as a latent variable, $q(x|y)$ is the variational posterior distribution, and the lower bound is denoted as $\mathcal{L}(y,z)$. By then taking a natural exponentiation of $\mathcal{L}(y,z)$, we get a lower bound of the marginal probability denoted as $\tilde{p}(y|z) = e^{\mathcal{L}(y,z)}$. Assume we are interested in plugging $p(y|z)$ into another probabilistic model $p(l|y)p(y|z)$ where y is a latent variable. With variational inference, the lower bound of the overall model can be derived as

$$\log p(l|z) \geq \int_y q(y) \log \frac{p(l|y)p(y|z)}{q(y)} \geq \int_y q(y) \log \frac{p(l|y)\tilde{p}(y|z)}{q(y)}, \tag{2}$$

where the usual variational lower bound is further lower bounded by replacing $p(y|z)$ with its lower bound $\tilde{p}(y|z)$. By substituting (1) into (2), we then derive the variational lower bound for the whole model,

$$\log p(l|z) \geq \int_{y,x} q(x|y,z)q(y|z) \log \frac{p(l|y)p(y|x)p(x|z)}{q(x|y,z)q(y|z)} = \mathcal{L}(l,z). \tag{3}$$

This example shows that variational inference has a recursive property that enables *inference modularity*. A technical challenge with VI is that the integral of the lower bound of a probabilistic module with respect to external latent variables, such as (2), may not always be tractable. Stochastic variational inference (SVI) offers an approximated solution to this new intractability by applying Monte Carlo Integration. Monte Carlo Integration is applicable to generic probabilistic distributions and lower bounds as long as we are able to draw samples from variational posterior.

In this case, the lower bound is approximated as

$$\mathcal{L}(l,z) \approx \frac{1}{N} \sum_i \log \frac{p(l|y_i)e^{\mathcal{L}(y_i,z)}}{q(y_i|z)}, \quad \mathcal{L}(y_i,z) \approx \frac{1}{M} \sum_j \log \frac{p(y_i|x_j)p(x_j|z)}{q(x_j|y_i,z)}, \tag{4}$$

where $y_i|z \sim q(y|z)$, $x_j|y_i,z \sim q(x|y_i,z)$ and N is the number of samples of y and M is the number of samples of x given y. Note that if there is a closed form solution of $\tilde{p}(y_i|z)$, the calculation of $\mathcal{L}(y_i,z)$ can be replaced with the closed-form solution.

MXFusion offers modularity via probabilistic modules, which combine the definition of probabilistic distributions and specialized inference methods in a concise interface. If using VI as the primary inference algorithm, probabilistic modules used in a model automatically compute variational lower bounds, such as $\tilde{p}(y|z)$ in the above example.

[2]https://github.com/amzn/MXFusion

```
m = Model()
m.x = Normal.define_variable(mean=0, variance=1, shape=(N, Q))
m.sigma2 = Variable(shape=(1,), transformation=PositiveTransformation())
m.y = SparseGaussianProcessRegression.define_variable(
    shape=(N, D), X=m.x, kernel=RBF(Q), noise_var=m.sigma2)
q = Posterior(m)
q.mu = Variable(shape=(N, Q))
q.S = Variable(shape=(N, Q), transformation=PositiveTransformation())
q.x.assign_factor(Normal(mean=q.mu, variance=q.S))
infr = GradientBasedInference(SVI(m, q, [m.x, m.y]))
```

Figure 1: Bayesian Gaussian process latent variable model (Titsias & Lawrence, 2010).

3 Example: Gaussian Process Latent Variable Model

A probabilistic module can be treated transparently as a probabilistic distribution. It is straightforward to construct a probabilistic model consists of multiple probabilistic modules such as deep Gaussian processes (GP) (Damianou & Lawrence, 2013; Dai et al., 2016; Salimbeni & Deisenroth, 2017). In these models, some of the exposed variables of probabilistic modules are latent variables. As shown in Section 2, as long as a variational inference method is used for the whole probabilistic model, the inference of individual probabilistic modules can be transparently handled. Figure 1 implements a simplified version of deep GP with only one layer, which is also called Bayesian Gaussian process latent variable model (BGPLVM) (Titsias & Lawrence, 2010), which is an example of this kind of models. BGPLVM can be constructed by assign the input variable $\mathbf{X}$ a Gaussian distribution with zero-mean and unit-variance. As the input variable $\mathbf{X}$ is a latent variable, the marginal log-likelihood is not tractable anymore. A variational lower bound can be written as

$$\log p(\mathbf{Y}) \geq \int_{\mathbf{X}} q(\mathbf{X}) \log \frac{p(\mathbf{Y}|\mathbf{X})p(\mathbf{X})}{q(\mathbf{X})} \geq \int_{\mathbf{X}} q(\mathbf{X}) \log \frac{\mathcal{L}_{\text{SGP}}(\mathbf{y}, \mathbf{X}, \mathbf{Z}, \theta)p(\mathbf{X})}{q(\mathbf{X})}$$

where $p(\mathbf{Y}|\mathbf{X})$ is a GP and $q(\mathbf{X}) = \mathcal{N}(\mathbf{X}|\mu, \text{diag}(\mathbf{s}))$ is the variational posterior of $\mathbf{X}$, which is assumed to be a Gaussian distribution with a diagonal covariance matrix. By applying a variational sparse GP (Titsias, 2009) approximation to $p(\mathbf{Y}|\mathbf{X})$, we further lower bound the original lower bound by replace $p(\mathbf{Y}|\mathbf{X})$ with the variational sparse GP lower bound $\mathcal{L}_{\text{SGP}}(\mathbf{y}, \mathbf{X}, \mathbf{Z}, \theta)$. For the expectation with respect to $q(\mathbf{X})$, we can apply stochastic variational inference (SVI) by drawing samples from $q(\mathbf{X})$. In this way, we result into a nested variational inference combining a generic inference method (SVI) with a specialized inference method (variational sparse GP). Following the same approach, it is also straight-forward to extend BGPLVM into a variationally auto-encoded GPLVM/deep GP (Dai et al., 2016) by parameterizing μ and $\mathbf{s}$ in $q(\mathbf{X})$ as the outcome of a DNN.

4 Conclusion

Current PPLs rely on generic inference methods, which maximizes the expressiveness and flexibility in terms of the family of probabilistic models that they can support. Those generic inference methods can be applied without knowing the formulation of a probabilistic model in advance, which makes it easy to achieve modularity in probabilistic models. However, this often results into a significant performance gap, which often makes it inapplicable to large scale real-world problems. Recent PPLs such as Edward (Tran et al., 2017) and Pyro (Goodman, 2017) have greatly reduced the performance gap by allowing users to customized posterior distributions that are, then, used by a generic variational inference. However, for sophisticated probabilistic models such as GP and Dirichlet process, state-of-the-art generic variational inference methods performs significantly worse than dedicated inference methods. In this paper, we propose to bridge this performance gap by incorporating those dedicated inference methods into PPLs in the form of probabilistic modules. A probabilistic module consists of both a model and inference definition, defined together and wrapped up in a modular, plug-and-play package. This allows specialized inference algorithms for corresponding probabilistic modules to be smoothly integrated into the inference algorithm of a larger probabilistic model.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`.

David M. Blei and Michael I. Jordan. Variational inference for dirichlet process mixtures. *Bayesian Anal.*, 1(1):121–143, 2006.

Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1), 2017.

Zhenwen Dai, Andreas Damianou, Javier González, and Neil Lawrence. Variational auto-encoded deep Gaussian processes. *International Conference on Learning Representations (ICLR)*, 2016.

Andreas Damianou and Neil Lawrence. Deep gaussian processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31, pp. 207–215, 2013.

Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: a language for flexible probabilistic inference. In *Proc. of AISTATS*, 2018.

Noah Goodman. Uber ai labs open sources pyro, a deep probabilistic programming language. `http://eng.uber.com/pyro`, 2017.

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In *Proc. of Uncertainty in Artificial Intelligence*, 2008.

J. Ishwaran and L. James. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96:161–174, 2001.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *J. Mach. Learn. Res.*, 18(1):430–474, 2017.

David J. Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. Winbugs - a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4): 325–337, 2000.

S. MacEachern. Estimating normal means with a conjugate style dirichlet process prior. *Communications in Statistics B*, 23:727–741, 1994.

T. Minka, J. M. Winn, J. P. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. Infer.net 2.6. Technical report, 2014.

Lawrence M. Murray. Bayesian state-space modelling on high-performance hardware using libbi. Technical report, 2013.

Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. Technical report, 2014.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Avi Pfeffer. Ibal: A probabilistic rational programming language. In *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 733–740, 2001.

Avi Pfeffer. Figaro: An object-oriented probabilis- tic programming language. *Charles River Analytics Technical Report*, 2009.

Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes. In *Advances in Neural Information Processing Systems 30*. 2017.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Michalis K. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *AISTATS*, 2009.

Michalis K. Titsias and Neil D. Lawrence. Bayesian gaussian process latent variable model. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. Deep probabilistic programming. In *International Conference on Learning Representations*, 2017.