Programming Languages for Deep Probabilistic Programming

Steven Lang
steven.lang.mz@gmail.com
TU Darmstadt

ABSTRACT

Probabilistic programming is an approach to adopt probabilistic models and inference as first-class citizens of a programming language. Its main goal is lowering the entry barrier into the field of probabilistic modeling and allow easier and faster prototyping in research.

In recent years, a new class of these languages has risen: Deep probabilistic programming languages. Their focus is on unifying probabilistic programming languages with the modeling power, efficiency, and composability of deep neural networks in the field of machine learning. We focus on Edward, a Turing-complete deep probabilistic programming language, and compare it to prior, as well as future work. Edward makes probabilistic programming as flexible and computationally efficient as deep learning and allows for rich compositions of probabilistic models and inference procedures.

KEYWORDS

machine learning, programming language, systems, probabilistic programming

1 INTRODUCTION

Probabilistic modeling is the task of describing the state of the world by using the mathematics of probability theory. With this, we can express all forms of uncertainty and noise associated with the model. We can make use of the inverse probability, or Bayes' rule, to answer questions about specific states in our world, given that we observe either everything else or only partial states from the rest of the world — this is called *probabilistic inference*. Using probabilistic modeling we can e.g. model the relationships between the time of the day, the day of the week, the traffic status between two cities and the weather conditions. This means we can now answer queries like *"What is the probability that there is a traffic jam on a Monday between 8 and 9 o'clock when it's rainy on my road to work?"* or *"At what time on Thursdays is the traffic the lowest?"*. We can either infer the probability of some state or use these probabilities to find the most likely state, given some observations of the world (called most probable explanation inference or *MPE*). Often, computing these probabilities exactly in probabilistic models is intractable (though there is a whole subclass of probabilistic models that focuses on tractable inference, such as Sum-Product Networks (Poon and Domingos, 2012) or Cutset Networks (Rahman et al., 2014)). Therefore, methods of approximate inference such as Markov Chain Monte Carlo, Variation Bayesian methods or expectation propagation have been developed, achieving a trade-off between computation time and accuracy.

Probabilistic programming concerns the syntax and semantics for programming languages, describing inference problems and constructing solvers that computationally characterize denoted probability distributions (van de Meent et al., 2018). Probabilistic programming is at the intersection and draws the attention of the machine learning, statistics and programming languages community. The last two decades have shown that the need for probabilistic programming languages (PPL) has increased and is long not satisfied as the following (incomplete) list of PPLs in chronological order shows: BUGS (Spiegelhalter et al., 1996), BNT (Murphy, 2001), IBAL (Pfeffer, 2001), JAGS (Plummer, 2003), BLOG (Milch et al., 2005), Figaro (Pfeffer, 2009), Church (Goodman et al., 2012), Augur (Tristan et al., 2013), LibBi (Murray, 2013), Venture (Mansinghka et al., 2014), Probabilistic-C (Paige and Wood, 2014), webPPL (Goodman and Stuhlmüller, 2014), PyMC (Salvatier et al., 2015), Anglican (Tolpin et al., 2015), BayesDB (Mansinghka et al., 2015), Hakaru (Narayanan et al., 2016), PSI (Gehr et al., 2016), CPProb (Casado et al., 2017), Stan (Carpenter et al., 2017), Birch (Murray and Schön, 2018), Touring.jl (Ge et al., 2018).

In recent years, the emerging field of deep learning has shown the importance of well established programming frameworks that back the foundations of the field. The nature of deep neural networks is compositional, that is, we can connect layers in creative ways and do neither need to worry about the actual forward propagation, nor about the backward propagation details based on gradient-based optimizations. Frameworks such as PyTorch (Paszke et al., 2019), TensorFlow (Abadi et al., 2015), CNTK (Seide and Agarwal, 2016) and MXNET (Chen et al., 2015) have accelerated the development of new successful deep learning architectures. With the rise of deep learning, new probabilistic modeling approaches, using deep neural networks as their building blocks, such as Variational Auto-Encoder (VAE) (Diederik et al., 2014), Normalizing Flows (Rezende and Mohamed, 2015) or Bayesian Recurrent Neural Networks (Fortunato et al., 2017) have surfaced. Since then, the next wave of PPLs has materialized: Deep probabilistic programming languages. These are built on top of already well established deep learning frameworks that are developed around a single key concept: computational graphs. Deep probabilistic programming languages use and extend this concept by introducing stochastic nodes representing probability distributions defined conditionally on their parent nodes. Tran et al. (2017) have leveraged these concepts in their novel deep probabilistic programming framework called Edward. Its main goal is to achieve the same composability and computational efficiency of deep learning for probabilistic modeling and probabilistic inference.

This paper is structured as follows: Section 2 mentions related work and the main trade-offs in deep probabilistic programming languages. Section 3 covers the contributions of Edward while Section 4 gives a critical view on its paper. Finally, Section 5 gives a short conclusion and outlook in the field of deep PPLs.

2 RELATED WORK

Probabilistic programming languages typically face the division into two groups, stemming from the trade-off between efficiency and expressiveness. PPLs such as BUGS (Spiegelhalter et al., 1996), BNT (Murphy, 2001), JAGS (Plummer, 2003), PyMC (Salvatier et al.,

2015) or Stan (Carpenter et al., 2017) are restricted to a niche class of probabilistic models and focus on optimizing the inference procedures for this specific subclass, gaining computational efficiency. Others like IBALAP (Pfeffer, 2001), Figaro (Pfeffer, 2009), BLOG (Milch et al., 2005) or Church (Goodman et al., 2012) are able to represent rich classes of graphical models, emphasizing expressiveness but suffer from more general inference procedure implementations and therefore do not scale well with increasing model or data size. Edward makes an effort to be the first PPL to bridge this gap: it is Turing-complete, supporting any computable probability distribution while implementing efficient inference algorithms, leveraging model structure and the computational graph.

Since then, deep probabilistic programming has also attracted large companies that support its development such as Pyro (Bingham et al., 2019) from Uber based on PyTorch and TensorFlow Probability (Dillon et al., 2017) from Google based on TensorFlow.

3 EDWARD: DEEP PROBABILISTIC PROGRAMMING

Edward is a deep probabilistic programming language. Probabilistic programming lets users specify probabilistic models as programs and compile those models down into inference procedures. At its core are two compositional representations as first-class citizens: random variables and inference. Edward allows fitting the same model using a variety of composable inference methods such as point estimation, variational inference, and Markov Chain Monte Carlo. Its key concept is to make no distinction between a model or an inference block, that is, models are simply a composition or collection of random variables while inference is the way of modifying parameters in this collection, subject to another.

To stay computationally efficient, Edward is based on TensorFlow and uses its computational graph. Therefore, it uses all benefits from TensorFlow like distributed model training, model parallelism, parameter vectorization and easy GPU support.

3.1 Compositional Representations for Models

One of Edward's focus is on the compositionality of probabilistic models. Therefore, Tran et al. have posed two main criteria on the compositional representation for probabilistic models. Firstly, Edward requires tight integration with computational graphs such that nodes can represent operations on the data while edges can represent data that is being communicated between the nodes. The second necessity is the invariance of model representations under the computational graph such that the graph can be reused during inference time. With these design principles, it becomes easy to develop probabilistic programs in a computational graph framework. It allows the composition of random variables to build arbitrary complex structures while still representing all operations in the computational graph itself.

To illustrate the power of Edward's compositionality, we demonstrate a simple Beta-Bernoulli model in Figure 1 which is described by the following joint probability distribution:

$$p\left(\mathbf{x}, \theta\right) = \text{Beta}\left(\theta \mid 1, 1\right) \prod_{n=1}^{50} \text{Bernoulli}\left(\mathbf{x} \mid \theta\right) \quad ,$$

Figure 1: The computational graph for a Beta-Bernoulli program. $\mathbf{x}^*$ is sampled from a Bernoulli distribution with $p = \theta^*$ and θ^* is sampled from a Beta distribution with parameters $a = 1, b = 1$.

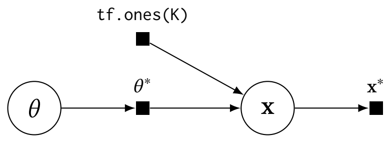
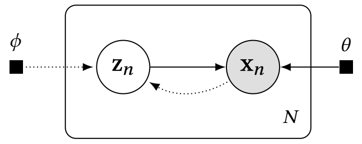


Figure 2: A Variational Auto-Encoder implementation constructed in Edward, with dotted lines for the inference model.



where $\mathbf{x} \in \mathbb{R}^{50}$ are 50 datapoints which all share the same latent variable θ, sampled from a beta distribution. The equivalent code in Edward is as simple as the following two lines:

```python
theta = Beta(a=1, b=1)
x = Bernoulli(p=tf.ones(50) * theta)
```

Evaluating the above snippet will evaluate the computational graph in Figure 1 at its root node: the sampling operation $\mathbf{x}^*$ from the specified Bernoulli distribution. For this operation to be evaluated, its parent must first be retrieved, that is, creating a 50-dimensional vector of ones using `tf.ones(50)` and performing the sampling operation on the beta distribution node to obtain a single θ value. Both of these operations have no dependencies and can thus be immediately computed. Finally, the sampling of $\mathbf{x}$ can be performed with θ and `tf.ones(50)` as input to obtain 50 Beta-Bernoulli distributed samples.

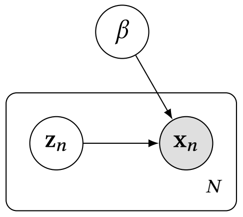
As an additional example, we want to highlight an implementation of Variational Auto-Encoder in Edward, visualized in Figure 2. The model is composed of a probabilistic model over the data distribution and a variational model to approximate the former's posterior.

```python
# Probabilistic model
z = Normal(mu=tf.zeros([N, d]), sigma=tf.ones([N, d]))
h = Dense(256, activation='relu')(z)
x = Bernoulli(logits=Dense(28 * 28, activation=None)(h))

# Variational model
qx = tf.placeholder(tf.float32, [N, 28 * 28])
qh = Dense(256, activation='relu')(qx)
qz = Normal(mu=Dense(d, activation=None)(qh),
            sigma=Dense(d, activation='softplus')(qh))
```

The program covers N data points $x_n \in \{0, 1\}^{28x28}$ with d latent variables $z_n \in \mathbb{R}^d$ each. Two densely connected layers with 256 hidden units using the ReLU activation function compose the probabilistic model, generating 28×28 binary pixel images. The second part defines the variational model with the same hidden layer architecture, producing output parameters of the normal posterior approximation. It becomes clear, that Edward allows for crisp and concise probabilistic programs: the probabilistic model, as well

Figure 3: Hierarchical model with local variables $\mathbf{z}_n$ and global variables β modeling the graph representation of Equation (1)



as the variational model can be easily extended, their probabilistic assumptions adapted.

Edward uses the higher-level framework Keras (Chollet et al., 2015) to include neural network building blocks such as the `Dense` layer used above.

3.2 Compositional Representations for Inference

To bridge the gap between the two groups of PPLs, Tran et al. pose two criteria on the compositional representation for inference. On the one hand, Edward shall support a large collection of different inference classes where the form of the inferred posterior depends on the inference algorithm. On the other hand, invariance of inference under the computational graph is desired, meaning the posterior can be composed as part of another model. This can be simply illustrated with the example in Figure 3 that represents a joint distribution over the data $\mathbf{x}$, local variables $\mathbf{z}$, and global variables β:

$$p(\mathbf{x}, \mathbf{z}, \beta) = p(\beta) \prod_{n=1}^{N} p(z_n \mid \beta)\, p(x_n \mid z_n, \beta) \quad . \tag{1}$$

In inference, we want to obtain the posterior distribution $p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta})$ with training data $\mathbf{x}_{train}$ and model parameters $\boldsymbol{\theta}$. This can be formulated as an optimization problem:

$$\min_{\boldsymbol{\lambda}, \boldsymbol{\theta}} \mathcal{L}\left(p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta}),\ q(\mathbf{z}, \beta; \boldsymbol{\lambda})\right) \quad , \tag{2}$$

where $q(\mathbf{z}, \beta; \boldsymbol{\lambda})$ is the distribution that approximates the true posterior $p(\mathbf{z}, \beta \mid \mathbf{x}_{train}; \boldsymbol{\theta})$, $\boldsymbol{\lambda}$ are model parameters of the posterior approximation, and $\mathcal{L}$ is a loss function between the true posterior p and its approximation q. The inference algorithm will determine the form of q, L and the rules to update the model parameters $\{\boldsymbol{\theta}, \boldsymbol{\lambda}\}$.

In Edward, this problem statement translates to the following code which defines and solves the optimization in Equation (2):

```
# Create inference with reference to variables
inference = ed.Inference(latent_vars={beta: qbeta, z: qz},
                         data={x: x_train})

# Build computational graph
inference.initialize()

# Run computations to update parameters
while condition:
    inference.update()
```

Table 1: Benchmarks comparing Edward against handwritten NumPy, Stan, PyMC3 and handwritten TensorFlow on logistic regression using Hamiltonian Monte Carlo iterations.

Probabilistic programming system	Runtime (s)
Handwritten NumPy (1 CPU)	534
Stan (1 CPU)	171
PyMC3 (12 CPU)	30.0
Edward (12 CPU)	**8.2**
Handwritten TensorFlow (GPU)	5.0
Edward (GPU)	**4.9**

where `qbeta` and `qz` are posterior variables and `x_train` are observed variables. With this design principle, Edward supports multiple sub-classes of inference procedures such as *Variational inference*, which uses a group of distributions to find a close approximation of the true posterior, *Monte Carlo inference*, which approximates the posterior using samples, and *Generative Adversarial Network inference*, mapping random points in a latent space to points in the data distribution.

Edward further allows posing inference as a collection of separate inference programs. As an example, the following code implements variational Estimation-Maximization (EM) inference (Neal and Hinton, 1993), using an approximate E-step (variational inference) over local and an M-step (MAP inference) over global variables:

```
# Define global and local variables
qbeta = PointMass(params=tf.Variable(tf.zeros([K, D])))
qz = Categorical(logits=tf.Variable(tf.zeros([N, K])))

# E-Step over local variables
inf_e = ed.VariationalInference(latent_vars={z: qz},
                                data={x: x_train, beta: qbeta})
# M-Step over global variables
inf_m = ed.MAP(latent_vars={beta: qbeta},
               data={x: x_train, z: qz})

# Expectation-Maximization loop
while not_converged:
  inf_e.update()  # Run E-Step
  inf_m.update()  # Run M-Step
```

3.3 Experiments

Tran et al. have benchmarked the runtime of Edward, Stan, PyMC, handwritten TensorFlow and handwritten NumPy using logistic regression on the Covertype dataset ($N = 581012, D = 54$) by performing a fixed number of Hamiltonian Monte Carlo (Neal, 2012) iterations. The benchmark results (see Table 1) show that Edward (GPU) delivers a 35x speedup over Stan (1 CPU) and a 6x speedup over PyMC3 (12 CPU). Moreover, Edward is as fast as handwritten TensorFlow code, implying, that no overhead is produced.

The hardware was a 12-core Intel i7-5930K CPU at 3.50GHz and an NVIDIA Titan X (Maxwell) GPU. It is important to note that the Stan implementation at the time of Tran et al.'s (2017) publication was only capable of using a single CPU. Starting with version `2.18` of Stan (`stan-math`), threading support can be switched on during compile-time (but is experimental and tagged unsafe). Furthermore,

Table 2: Different inference procedures for a probabilistic encoder-decoder architecture evaluating the negative log-likelihood on the binarized MNIST dataset, showing that Edward makes it easy to develop and experiment with a large pool of inference algorithms.

Inference method	Negative log-likelihood
VAE (Diederik et al., 2014)	≤ 88.2
VAE without analytic KL	≤ 89.4
VAE with analytic entropy	≤ 88.1
VAE with score function gradient	≤ 87.9
Normalizing flows (Rezende and Mohamed, 2015)	≤ 85.8
Hierarchical variational model (Ranganath et al., 2015)	≤ 85.4
Importance-weighted auto-encoders ($K = 50$) (Burda et al., 2015)	≤ 86.3
HVM with IWAE objective ($K = 5$)	≤ 85.2
Rényi divergence ($\alpha = -1$) (Li and Turner, 2016)	≤ 140.5

even though PyMC3 uses Theano (Al-Rfou et al., 2016) as a backend, its GPU version was slower than the 12-CPU version, largely due to communication overhead with NumPy.

To demonstrate Edward's flexibility, Tran et al. have implemented an array of experiments of complex inference algorithms (see Table 2) with a probabilistic encoder-decoder architecture and evaluated them using held-out log-likelihoods on the binarized MNIST dataset (Salakhutdinov and Murray, 2008).

4 DISCUSSION

With Edward being a programming framework and its paper delivering many code examples, it was natural to go forth and try it out ourselves. Unfortunately, the development of Edward has stopped in July 2018. This resulted in issues such as incompatible dependencies with Edward, as well as conflicting dependencies. Neither code examples from the paper Appendix, nor tutorial Jupyter notebooks from Edward's GitHub repository [1] continue to work. Therefore, no kind of reproducibility could be achieved in our experiments. Edward was dropped in favor of Edward2 (Tran et al., 2018), its direct successor, which was adopted by Google. Edward2 has further achieved inclusion as a sub-module into TensorFlow Probability, a successor of the earlier work on TensorFlow Distributions (Dillon et al., 2017).

Tran et al. (2017) suggest in their related work section, that Edward bridges the gap between PPLs which focus on efficiency but sacrifice generality and PPLs which focus a rich class of covered models but come with decreased performance that does not scale well. In the following sections the author make good points on how they have achieved the covering of a broad range of possible models due to their model and inference composability. Unfortunately, the scaling with respect to model size, as well as data size and complexity has not been empirically shown and is therefore only proposed.

Similar to PyTorch and TensorFlow, Edward has also promised a so-called "model zoo" with its release[2]. A model zoo in the field of deep learning is known as a repository of parametric model architectures with fixed parameters provided that have been extensively optimized on a specific problem statement. The idea is that other users can then simply download and skip the expensive optimization process to use the model in a plug-and-play fashion and compose it with other parts of their own modeling structure. As of this time (February 2020), the Zoo still seems to be unavailable without any redirection or information on alternatives. It seems that this has been a problem in the past, mentioned by reviewers on OpenReview[3]. The associated GitHub issue regarding Edward's roadmap has an item for its model zoo which to this day still states that it is not yet complete.

Furthermore, code examples in Edward were hard to comprehend and follow since they were only snippets extracted from a larger codebase. The amount of code left out remained was not clear. Therefore, even though code examples were provided with good intentions, they have further confused the reader due to being out of context snippets.

Another important criterion for new PPLs (and serious programming languages in general) is their applicability in real-world use-cases. Code samples were mostly given but not empirically evaluated and compared against previous implementations in other frameworks. The authors have reduced their empirical study to an evaluation of Variational Auto-Encoder with a set of different inferences techniques that were already easily accessible in pure TensorFlow.

5 CONCLUSION AND OUTLOOK

The Edward library offers a large collection of probabilistic models and inference procedures in an easily usable form. Its main contributions are the power of compositional representations of probabilistic models and probabilistic inference procedures. To bridge the gap between efficient and general-purpose PPLs, Edward is based on TensorFlow's static computation graph and therefore leverages the advantages of fast, parallelizable computations and GPU support, which in theory should scale to large model and data sizes. A major drawback of Edward is the lack of support for dynamically changing computation graphs that have recently been adopted in TensorFlow, starting with version 2. This has been acknowledged by the authors and addressed in its successor, Edward2.

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. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermüller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul F. Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron C. Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Melanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian J. Goodfellow, Matthew Graham, Çaglar Gülçehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrançois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro,

[1]https://github.com/blei-lab/edward/tree/master/notebooks
[2]http://edwardlib.org/zoo
[3]https://openreview.net/forum?id=Hy6b4Pqee&noteId=Sy0_4dsGl

Robert McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Joseph Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph P. Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. 2016. Theano: A Python framework for fast computation of mathematical expressions. *CoRR* abs/1605.02688 (2016). arXiv:1605.02688 http://arxiv.org/abs/1605.02688

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *J. Mach. Learn. Res.* 20 (2019), 28:1–28:6. http://jmlr.org/papers/v20/18-403.html

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. 2015. Importance Weighted Autoencoders. (09 2015).

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software, Articles* 76, 1 (2017), 1–32. https://doi.org/10.18637/jss.v076.i01

Mario Lezcano Casado, Atilim Gunes Baydin, David Martínez-Rubio, Tuan Anh Le, Frank D. Wood, Lukas Heinrich, Gilles Louppe, Kyle Cranmer, Karen Ng, Wahid Bhimji, and Prabhat. 2017. Improvements to Inference Compilation for Probabilistic Programming in Large-Scale Scientific Simulators. *CoRR* abs/1712.07901 (2017). arXiv:1712.07901 http://arxiv.org/abs/1712.07901

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR* abs/1512.01274 (2015). arXiv:1512.01274 http://arxiv.org/abs/1512.01274

François Chollet et al. 2015. Keras. https://keras.io.

P Kingma Diederik, Max Welling, et al. 2014. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Vol. 1.

Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous. 2017. TensorFlow Distributions. *CoRR* abs/1711.10604 (2017). arXiv:1711.10604 http://arxiv.org/abs/1711.10604

Meire Fortunato, Charles Blundell, and Oriol Vinyals. 2017. Bayesian Recurrent Neural Networks. *CoRR* abs/1704.02798 (2017). arXiv:1704.02798 http://arxiv.org/abs/1704.02798

Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: a language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain.* 1682–1690. http://proceedings.mlr.press/v84/ge18b.html

Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs, Vol. 9779. 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2012. Church: a language for generative models. *CoRR* abs/1206.3255 (2012). arXiv:1206.3255 http://arxiv.org/abs/1206.3255

Noah D Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org. Accessed: 2020-2-27.

Yingzhen Li and Richard E. Turner. 2016. Rényi Divergence Variational Inference. arXiv:stat.ML/1602.02311

Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. 2014. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR* abs/1404.0099 (2014). arXiv:1404.0099 http://arxiv.org/abs/1404.0099

Vikash K. Mansinghka, Richard Tibbetts, Jay Baxter, Patrick Shafto, and Baxter Eaves. 2015. BayesDB: A probabilistic programming system for querying the probable implications of data. *CoRR* abs/1512.05006 (2015). arXiv:1512.05006 http://arxiv.org/abs/1512.05006

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel Ong, and Andrey Kolobov. 2005. BLOG: Probabilistic Models with Unknown Objects. *IJCAI* 2005 (01 2005).

Kevin P. Murphy. 2001. The Bayes Net Toolbox for MATLAB. *Computing Science and Statistics* 33 (2001), 2001.

Lawrence M. Murray. 2013. Bayesian State-Space Modelling on High-Performance Hardware Using LibBi. arXiv:stat.CO/1306.3277

Lawrence M. Murray and Thomas B. Schön. 2018. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control* 46 (2018), 29–43. https://doi.org/10.1016/j.arcontrol.2018.10.013

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic inference by program transformation in Hakaru (system description). In *International Symposium on Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings.* Springer, 62–79. https://doi.org/10.1007/978-3-319-29604-3_5

Radford Neal and Geoffrey Hinton. 1993. A New View of the EM Algorithm that Justifies Incremental and Other Variants. (03 1993).

Radford M. Neal. 2012. MCMC using Hamiltonian dynamics. arXiv:stat.CO/1206.1901

Brooks Paige and Frank D. Wood. 2014. A Compilation Target for Probabilistic Programming Languages. *CoRR* abs/1403.0504 (2014). arXiv:1403.0504 http://arxiv.org/abs/1403.0504

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Avi Pfeffer. 2001. IBAL: A Probabilistic Rational Programming Language. In *IJCAI*.

Avi Pfeffer. 2009. Figaro : An Object-Oriented Probabilistic Programming Language.

Martyn Plummer. 2003. JAGS: A Program for Analysis of Bayesian Graphical Models using Gibbs Sampling. *3rd International Workshop on Distributed Statistical Computing (DSC 2003); Vienna, Austria* 124 (04 2003).

Hoifung Poon and Pedro Domingos. 2012. *Sum-Product Networks: A New Deep Architecture.* arXiv:cs.LG/1202.3732v1

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. 2014. Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees. 630–645. https://doi.org/10.1007/978-3-662-44851-9_40

Rajesh Ranganath, Dustin Tran, and David M. Blei. 2015. Hierarchical Variational Models. arXiv:stat.ML/1511.02386

Danilo Jimenez Rezende and Shakir Mohamed. 2015. Variational Inference with Normalizing Flows. arXiv:stat.ML/1505.05770

Ruslan Salakhutdinov and Iain Murray. 2008. On the quantitative analysis of deep belief networks. *Proceedings of the 25th International Conference on Machine Learning*, 872–879. https://doi.org/10.1145/1390156.1390266

John Salvatier, Thomas Wiecki, and Christopher Fonnesbeck. 2015. Probabilistic Programming in Python using PyMC. arXiv:stat.CO/1507.08050

Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 2135. https://doi.org/10.1145/2939672.2945397

David Spiegelhalter, Andrew Thomas, Nicky Best, and Wally Gilks. 1996. BUGS 0.5: Bayesian inference using Gibbs sampling manual (version ii). *MRC Biostatistics Unit, Institute of Public Health, Cambridge, UK* (1996), 1–59.

David Tolpin, Jan-Willem Meent, and Frank Wood. 2015. Probabilistic Programming in Anglican. 308–311. https://doi.org/10.1007/978-3-319-23461-8_36

Dustin Tran, Matthew D. Hoffman, Dave Moore, Christopher Suter, Srinivas Vasudevan, Alexey Radul, Matthew Johnson, and Rif A. Saurous. 2018. Simple, Distributed, and Accelerated Probabilistic Programming. In *Neural Information Processing Systems.*

Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. 2017. Deep Probabilistic Programming. arXiv:stat.ML/1701.03757

Jean-Baptiste Tristan, Daniel Huang, Joseph Tassarotti, Adam Pocock, Stephen J. Green, and Guy L. Steele Jr. 2013. Augur: a Modeling Language for Data-Parallel Probabilistic Inference. arXiv:stat.ML/1312.3613

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. *An Introduction to Probabilistic Programming.* arXiv:stat.ML/1809.10756v1