ForneyLab.jl: Fast and flexible automated inference through message passing in Julia *

Marco Cox
m.g.h.cox@tue.nl
Eindhoven University of Technology
Eindhoven, the Netherlands

Thijs van de Laar
t.w.v.d.laar@tue.nl
Eindhoven University of Technology
Eindhoven, the Netherlands

Bert de Vries
bdevries@ieee.org
Eindhoven University of Technology &
GN Hearing, Eindhoven (NL)

1 Introduction

To accommodate automatic (Bayesian) inference in a large class of models, probabilistic programming systems usually rely on inference methods that require no manual derivations. Most of these methods can be categorized into two families: (i) MCMC methods, which generate samples from the exact posterior distribution, and (ii) variational methods, which aim to optimize the parameters of a tractable approximation to the true posterior. To avoid model-specific derivations, variational methods in this setting are often based on stochastic optimization, yielding algorithms such as black-box variational inference (BBVI) [8] and automatic differentiation variational inference (ADVI) [4].

While these 'derivation-free' algorithms can perform inference in a wide range of models, their generality comes at the price of significant computational load. In some cases, this price is inherent to the model at hand: we might simply not be able to construct an algorithm that can perform inference at a significantly lower computational cost. However, there is also a large family of practical models (like the random walk model we will discuss in this paper) for which more efficient inference algorithms can be devised by exploiting model-specific structures and properties. For example, an efficient algorithm might leverage closed-form solutions to integrals or local approximations to speed up inference. Ideally, such an efficient algorithm would also be generated automatically by a probabilistic programming framework, yielding a faster algorithm without additional manual work.

While fully automatic generation of inference algorithms that exploit model-specific properties might not be possible, the message passing paradigm provides a convenient way to generate such algorithms semi-automatically. By representing the probabilistic model as a factor graph, inference tasks on the model can be decomposed into a collection of local inference tasks. These local inferences can be interpreted as messages flowing over the graph. Importantly, each message involves only node-local computations. Thus, it is possible to build a library of pre-derived solutions to local computations (message updates) for a collection of factor nodes. These pre-derived messages can then be reused across models. Since the process of combining local messages into a global inference algorithm can be automated, it is possible to automatically generate inference algorithms for models composed of nodes that are covered by the library. Examples of inference methods that fit this paradigm include: (loopy) belief propagation for exact inference, mean-field and structured variational message passing (VMP) [10], and expectation propagation (EP) [7] for approximate inference.

In this paper we present ForneyLab.jl[1], an open-source message passing-based probabilistic programming toolbox written in Julia [1].

2 ForneyLab.jl: A Julia toolbox for message passing-based inference

We developed ForneyLab.jl to automatically generate code that implements (approximate) Bayesian inference in a given model through message passing. It consists of the following elements:

- A convenient syntax for specifying probabilistic models.
- A library of common factor nodes and corresponding message computations, and a convenient system to define custom nodes and message updates.
- Automatic message passing algorithm generators (currently: belief propagation, VMP and EP).
- A code generator that converts constructed algorithms into executable Julia code (that can be modified and debugged).

ForneyLab was designed with a focus on flexibility: it provides convenient ways to add custom factor nodes, message update rules and entire message passing algorithms. Moreover, the user is free to inspect and modify intermediate results in the code generation pipeline, for example by customizing part of the message passing schedule. Internally, ForneyLab represents the model as a *Forney-style* factor graph [5], which is a conducive framework when working with time series models such as dynamical state-space models.

Automated inference through message passing and a library of efficient message computations is already available through Infer.NET [6]. However, in contrast to Infer.NET, ForneyLab provides an fully open and extensible (Julia) code base for algorithm generation. Rather than generating a binary that performs inference, ForneyLab explicitly generates source code that can be inspected and modified. The Julia language, with its MATLAB-like syntax and meta-programming functionalities, marries the productivity of a high-level language with high run-time performance.

3 Example: fitting a random walk model

As an example, we use ForneyLab to perform full Bayesian inference in a random walk model with noisy observations:

$$x_t \,|\, x_{t-1}, d, w \sim \mathcal{N}(x_{t-1} + d, w^{-1}),$$
$$y_t \,|\, x_t, u \sim \mathcal{N}(x_t, u^{-1}).$$

Our goal is to approximate the posterior distribution of the hidden state sequence $[x_1, \ldots, x_T]$ and model parameters $\{d, w, u\}$ from observations $[y_1, \ldots, y_T]$ under appropriate priors (see Fig. 1 for details). A condensed version of the code required to specify this model, build a variational Bayesian inference algorithm, and perform the actual inference with ForneyLab is listed in Fig. 1. A snippet of the algorithm code generated by ForneyLab is listed in Fig. 2.

We test the algorithm by applying it on an observation sequence ($T = 50$) obtained by sampling the generative model under fixed parameters. To evaluate the quality of the inference result, we calculate the average log-likelihood of a test set under the posterior

*To be presented at ProbProg 2018.

[1] ForneyLab.jl is available at
https://github.com/biaslab/ForneyLab.jl

```
# Specify the generative model
@RV x_0 ~ GaussianMeanVariance(0.0, 100.0) # State prior
@RV d ~ GaussianMeanVariance(0.0, 100.0) # Drift prior
@RV w ~ Gamma(0.01, 0.01) # State transition precision prior
@RV u ~ Gamma(0.01, 0.01) # Observation precision prior
x_t_min = x_0
for t = 1:T
    @RV z[t] ~ GaussianMeanPrecision(d, w)
    @RV x[t] = x_t_min + z[t] # State transition
    @RV y[t] ~ GaussianMeanPrecision(x[t], u) # Observation
    placeholder(y[t], :y, index=t) # Data placeholder
    x_t_min = x[t] # Ref to previous state
end

# Generate a VMP schedule and compile to Julia code
q = RecognitionFactorization([x_0; x; z], d, w, u,
  ids=[:X, :D, :W, :U]) # Specify a q-factorization
algo = variationalAlgorithm(q) # Yields string containing code
eval(parse(algo)) # Load algorithm in local scope

# Execute the inference algorithm
data = Dict(:y => y_data)
marginals = Dict{Symbol, ProbabilityDistribution}(...)
for i = 1:n_its
    stepX!(data, marginals)
    stepW!(data, marginals)
    stepU!(data, marginals)
    stepD!(data, marginals)
end
```

Figure 1: Code snippet to specify a random walk model, generate a (structured variational Bayes) inference algorithm and execute it. `@RV` is a macro for conveniently defining random variables.

predictive distribution. The test set contains 1000 random continuations of the training sequence for 20 time steps. Fig 3 shows the results ("fl-svb") as a function of the algorithm iteration count, and places ForneyLab within the speed-performance landscape relative to a variety of more generic (black-box) inference algorithms. Since the algorithm generated by ForneyLab leverages pre-derived message computations, execution is faster than black-box inference algorithms based on sampling or stochastic optimization. However, since these message computations are not model-specific, generating the algorithm with ForneyLab does not require any additional work from the user compared to more generic probabilistic programming systems. We argue that this is an interesting proposition in many practical settings, such as real-time probabilistic signal processing, learning from streaming data, and probabilistic data processing on embedded devices.

An elaborate discussion and evaluation of ForneyLab is forthcoming [3].

References

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, Jan. 2017.

[2] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 76(1), 2017.

[3] M. Cox, T. van de Laar, and B. de Vries. A Factor Graph Approach to Automated Design of Bayesian Signal Processing Algorithms. *Under review*.

[4] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei. Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18(1):430–474, 2017.

[5] H.-A. Loeliger, J. Dauwels, J. Hu, S. Korl, L. Ping, and F. R. Kschischang. The Factor Graph Approach to Model-Based Signal Processing. *Proceedings of the IEEE*, 95(6):1295–1322, June 2007.

[6] T. Minka, J. Winn, J. Guiver, Y. Zaykov, D. Fabian, and J. Bronskill. Infer.NET 2.7, 2018. Microsoft Research Cambridge.

[7] T. P. Minka. Expectation Propagation for Approximate Bayesian Inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[8] R. Ranganath, S. Gerrish, and D. Blei. Black Box Variational Inference. In *PMLR*, pages 814–822, Apr. 2014.

[9] D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

[10] J. Winn and C. M. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(Apr):661–694, 2005.


```
function stepX!(data::Dict, marginals::Dict=Dict(),
  messages::Vector{Message}=Array{Message}(349))
    ...
    messages[4] = ruleSPAdditionOutVGG(nothing, messages[3],
      messages[2])
    messages[5] = ruleVBGaussianMeanPrecisionM(
      ProbabilityDistribution(Univariate, PointMass,
      m=data[:y][1]), nothing, marginals[:u])
    ...
    marginals[:x_0] = messages[3].dist * messages[349].dist
    marginals[:x_1] = messages[4].dist * messages[346].dist
    ...
    return marginals
end
```

Figure 2: Snippet of the algorithm code generated by ForneyLab to update the approximate posterior on the hidden state variables. Every recognition factor in the variational approximation has its own "step" function. Moreover, if the variational free energy can be evaluated analytically, ForneyLab can also generate a function for that.

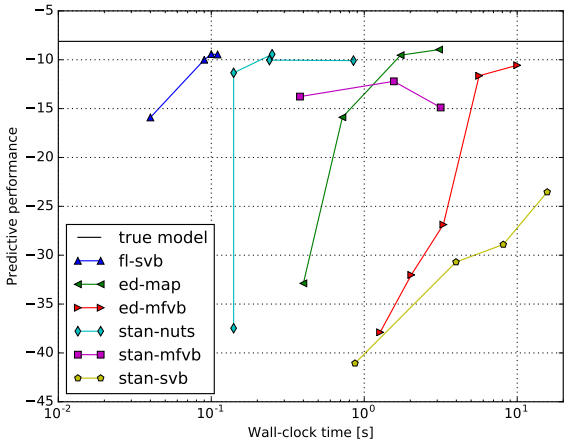

Figure 3: Average posterior log-likelihood of the test set as a function of running time for multiple inference methods and toolboxes. Markers correspond to different iteration counts or posterior samples under the respective implementations. Methods `ed-map` and `ed-mfvb` correspond to MAP and mean-field variational Bayes implemented in Edward [9] (without GPU acceleration). The `stan-*` labels correspond to probabilistic programming framework Stan [2].