Philipp Gabler, BSc

Automatic Graph Tracking in Dynamic Probabilistic Programs via Source Transformations

Master's Thesis

to achieve the university degree of
Master of Science

submitted to
Graz University of Technology

Supervisor
Univ.-Prof. Dipl.-Ing. Dr. mont. Franz Pernkopf

Co-supervisor
Dipl.-Ing. Martin Trapp, BSc

Institute of Signal Processing and Speech Communication

Faculty of Electrical and Information Engineering

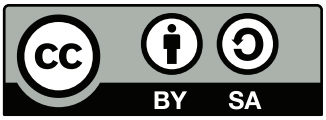
Graz, XXXX 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature



The LaTeX source of this document is available at
https://github.com/phipsgabler/master-thesis
or upon request from the author.[1]

[1] pgabler@student.tugraz.at

Abstract

This thesis presents a novel approach for the implementation of a tracking system to facilitate program analysis, based on program transformations. The approach is then applied to a specific problem in the field of probabilistic programming.

The main contribution is a general system for the extraction of rich computation graphs in the Julia programming language, based on a transformation of the intermediate representation (IR) used by the compiler. These graphs contain the whole recursive structure of any Julia program in terms of executed IR instructions. The system is flexible enough to be used for multiple purposes that require dynamic program analysis or abstract interpretation, such as automatic differentiation or dependency analysis.

The second part of the thesis describes the application of this graph tracking system to probabilistic programs written for Turing, a probabilistic programming system implemented as an embedded language within Julia. Through this, an executed Turing model can be analyzed, and the dependency structure of involved random variables be extracted from it. Given this structure, analytical Gibbs conditionals can be calculated and passed to Turing's inference mechanism, where they are used in Markov-Chain Monte Carlo samplers approximating the modelled distribution.

Contents

Notation

$\mathbb{P}[\Theta \in A \mid X = x]$	Random variables and their realizations will usually be denoted with upper and lower case letters, respectively (with some exceptions for Greek variable names). Sets are written with uppercase letters.
$\mathbb{E}[X], \mathbb{V}_X[f(X, Y)]$	Expectation and variance; if necessary, the variable with respect to which the moment is taken is indicated.
$\phi(x), f_Z(x)$	Density functions are named using letters commonly used for functions, with an optional subscript indicating the random variable they belong to. Densities always come with implied base measures depending on the type of the random variable.
$p(x, y \mid z)$	The usual abuse of notation with the letter "p" standing for any density indicated by the names of the variables given to it is used when no confusion arises (in this case, $f_{X,Y\mid Z}$ is implied).
$y \mapsto p(x \mid y, z)$	Anonymous functions are distinguised from function evaluation; this is crucial to differentiate between probability densities and likelihoods, for example.
$\int p(x)\,\mathrm{d}x = 1$	Integrals over the whole domain of a density are written as indefinite integrals, where the usage is clear.
$[x, y, z] = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$	For consistency with Julia code, vectors (arrays of rank 1) are written in brackets. Thereby, the form written in a row denotes a column vector; actual row vectors are written as transposed column vectors.
$\Theta^{(k)} = [\Theta_1^{(k)}, \ldots, \Theta_N^{(k)}]$	Superscript indices in parentheses are used for series or sequences of variables, and subscript indices for components of multivariate variables.
$z_{-i} = [z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_N]$	Negative indices denote all components of a variable without the negated one.

$f.(x, 1) = [f(x_1, 1), \ldots, f(x_N, 1)]$

Functions with a period indicate vectorized application, as in Julia code[2]: the function is applied over all elements of the input arrays individually, whereby arrays of lower rank or scalars are "broadcasted" along dimensions as necessary.

[2] See https://docs.julialang.org/en/v1/manual/functions/#man-vectorized-1

1 Introduction

The idea of this work has already been described in [Gab+19].

1.1 Problem Description

1.2 Related Work

2 Background

This section provides the background for the concepts used later in chapters 3 and 4. Initially, it gives a quick overview of Baysian inference and probabilistic programming in general, necessary to understand the requirements and usual approaches of probabilistic programming systems.

Consequently, the machinery and language used to develop the graph tracking system forming the main part of the work are described. This consists firstly of a short introduction to graph tracking and source-to-source automatic differentiation, which contain many ideas and terminology that will be used later, and often provided inspiration. Secondly, the basic notions and techniques of the Julia compilation process as well as the language's metaprogramming capabilities are described, which form the basis of the implementation.

2.1 Bayesian Inference and MCMC methods

Generative modelling is a technique for modelling phenomena based on the assumption that observables can be fully described through some stochastic process. When we assume this process to belong to a specified family of processes, the estimation of the "best" process is a form of learning: if we have a good description of how obserations are generated, we can make summary statements about the whole population (descriptive statistics) or predictions about new observations. When observations come in pairs of independent and dependent variables, learning the conditional model of one given the other solves a regression or classification problem.

Within a Baysian statistical framework, we assume that the family of processes used is specified by random variables related through conditional distributions with densities, which describe how the observables would be generated: some *unobserved variables* are generated from *prior distributions*, and the *observed data* are generated conditionally on the unobserved variables. The goal is to learn the *posterior distribution* of the parameters given the observations, which is a sort of "inverse" of how the problem is specified.

As an example, consider image classification: if we assume that certain percentages of an image data set picture cats and dogs, respectively, the distribution of these labels forms the prior. Given the information which kind of animal is depicted on it, an image can then be generated as a matrix of pixels based on a distribution of images conditioned on labels. The posterior distribution is then conditional distribution of

the label given an image. When we have this information, we can, for example, build a Baysian classifier, by returning for a newly observed image that label which has the higher probability under the posterior.

This kind of learning is called Bayesian inference since, in the form of densities, the form of the model can be expressed using Bayes' theorem as the conditional distribution[1]

$$\overbrace{p(\theta \mid x)}^{\text{posterior}} = \frac{\overbrace{p(x \mid \theta)}^{\text{likelihood}} \overbrace{p(\theta)}^{\text{prior}}}{p(x)}, \tag{2.1}$$

where x are the observed data, and θ are the unobserved parameters. The posterior represents the distribution of the unobserved variables as a combination of the prior belief updated by what has been observed [Cono6]. (In practice, not all of the unobserved variables have to be model parameters we are actually interested in; these can be integrated out).

Going beyond simple applications like the classifier mentioned above, handling the posterior gets difficult, though. Simply evaluating the posterior density $\theta \mapsto p(\theta \mid x)$ at single points is not enough for usages such as prediction, parameter estimation, or evaluation of probabilities of continuous variables. The problem is that almost all of the relevant quantities depend on some sort of expectation over the posterior density, an integral of the form

$$\mathbb{E}[f(\Theta) \mid X = x] = \int f(\theta) p(\theta \mid x) \, \mathrm{d}\mu(\theta), \tag{2.2}$$

(with the base measure μ depending on the type of Θ). This in turn involves calculating the normalizing marginal

$$p(x) = \int p(x, \tilde{\theta}) \, \mathrm{d}\mu(\tilde{\theta}). \tag{2.3}$$

in equation 2.1 (also called the "evidence").

When the distributions involved form a sufficiently "nice" combination, e.g., a conjugate pair, the integration can be performed analytically, since the posterior density has a closed form for a certain known distribution, or at least is a known integral. In general, however, this is not intractable, not even numerically, and approximations have to be made. Even for discrete variables, combinatorial explosion limits the applicability of simple summation.

ref

DIFFERENT TECHNIQUES for posterior approximation are available: among them are distribution-based methods like message passing and variational inference for general graphical models. The methods described in this thesis, however, apply Monte Carlo methods, a sampling-based approach. Their basic idea is to derive, from

ref

[1] Note the abuse of notation regarding $p(\cdot)$; see page xi on notation.

a specified density, a sampling procedure with a consistent estimator for expectations:

$$I^{(k)}(f) \to \mathbb{E}[f(\Theta) \mid X = x], \quad \text{as} \quad k \to \infty \tag{2.4}$$

in some appropriate stochastic convergence (usually convergence in probability is enough) [Vih20]. Most of these methods are defined in a form that samples a sequence of individual random variables $\Theta^{(k)}$, called a *chain*, for which a law of large numbers (LLN) holds:

$$I^{(k)}(f) = \frac{1}{n}\sum_{i=1}^{n} f(\Theta^{(k)}) \to \mathbb{E}[f(\Theta) \mid X = x] \tag{2.5}$$

When we can sample $\Theta^{(k)} \sim p(\cdot \mid x)$ exactly, they are i.i.d. and the LLN holds trivially; such samplers exist, but might also be difficult to derive or not possess good enough convergence properties. Another large class of samplers is formed by *Markov Chain Monte Carlo* (MCMC) methods, which, instead of sampling exactly from the density, define $\Theta^{(k)}$ via a Markov chain: by choosing the transition the right way, the resolving Markov chain is ergodic with the target density as the unique stationary distribution. The advantage of MCMC methods is that they mostly treat the density as a black box, without requiring any more formal manipulations on its structure.

overthink advantages

Frequently, MCMC methods use variations of the Metropolis-Hastings algorithm (MH), which requires the definition of a Markov transition kernel by means of two helper fuctions: a proposal distribution with density q, and an acceptance rate α, both depending on the old value in the chain:

make an algorithm

1. Start from an arbitrary $\Theta^{(1)}$.
2. For each $k \geq 1$:
    1. Propose $\hat{\Theta}^{(k)} \sim q(\Theta^{(k-1)}, \cdot)$.
    2. With probability $\alpha(\hat{\Theta}^{(k)}, \Theta^{(k-1)})$, set $\Theta^{(k)} = \hat{\Theta}^{(k)}$; else, keep $\Theta^{(k)} = \Theta^{(k-1)}$.

There exist many MH-based schemes with different properties and requirements: the classical random-walk Metropolis algorithm with Gaussian proposals, Reversible Jump MCMC, or gradient-informed methods like Metropolis Adjusted Langevin and Hamiltonian Monte Carlo (HMC).

cite all that

When we have a multi-component structure $\Theta = [\Theta_1, \ldots, \Theta_N]$, a full transition kernel can be hard to find, and we can instead use a family of componentwise updates, given by conditional kernels q_i operating on only one component of Θ, with the others fixed:

$$\begin{aligned} \hat{\Theta}^{(k)}_{-i} &= \Theta^{(k-1)}_{-i} \\ \hat{\Theta}^{(k)}_{i} &\sim q_i(\Theta^{(k-1)}_{i}, \cdot \mid \Theta^{(k-1)}_{-i}) \end{aligned} \tag{2.6}$$

Components can be scalars or multivariate blocks, and the kernel may itself be any valid transition kernel. This allows to freely mix different MCMC methods.

This "within-Gibbs" sampler bears its name because it is a generalization of the classical Gibbs sampling algorithm: the conditional densities $\Theta_i \mapsto p(\Theta_i \mid \Theta_{-i}, x)$ can directly be used as component proposals for a within-Gibbs sampler, leading to a cancelling acceptance rate of $\alpha \equiv 1$. This sampler has the advantage that it is in many cases rather easy to derive, even manually, from a given factorization of the joint density.

2.2 Probabilistic Programming

Probabilistic programming is a means of describing probabilistic models through the syntax of a programming language. Probabilistic programms distinguish themselves from normal programs by the possibility of being sampled from conditionally, with some of the internal variables fixed to observed values. While probabilistic programming systems are often implemented as separate, domain-specific languages, they can also be embedded into "host" programming languages with sufficient syntactic flexibility. The latter is advantageous if one wants to use regular general-purpose programming constructs or interact with other functionalities of the host language.

2.3 Computation Graphs and Automatic Differentiation

2.4 Metaprogramming and Compilation in Julia

3 Implementation of Dynamic Graph Tracking in Julia

3.1 Automatic Graph Tracking and Extended Wengert Lists

4 Graph Tracking in Probabilistic Models

5 Discussion

5.1 Future Work

Bibliography

[Cono6] Congdon, P. *Bayesian statistical modelling*. 2nd ed. Wiley Series in Probability and Statistics. Chichester, England ; Hoboken, NJ: John Wiley & Sons, 2006. 573 pp.

[Gab+19] Gabler, P. et al. "Graph Tracking in Dynamic Probabilistic Programs via Source Transformations". In: 2nd Symposium on Advances in Approximate Bayesian Inference. Vancouver, 2019-10-16. URL: https://openreview.net/forum?id=r1eAFknEKr (visited on 2020-07-07).

[Vih20] Vihola, M. *Lectures on stochastic simulation*. University of Jyväskylä, 2020. URL: http://users.jyu.fi/~mvihola/stochsim/.

Colophon

This document was typeset using the pdfLaTeX typesetting system, with the memoir document class. The body text is set in 11 pt Linux Libertine, enhanced by the microtype package. Other fonts include Biolinum and Inconsolata.

The document source has been written in Emacs with AUCTeX mode, using TeXworks as PDF viewer.