Working Towards Distributed Inference Compilation at Scale

Frank Wood
fwood@cs.ubc.ca
Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model

Frank Wood
fwood@cs.ubc.ca
Outline

• Probabilistic Programming
• Model-Based Reasoning & Inference
• Inference Compilation
• The Quest for New Physics
• Challenges
• Massively Distributed Deep Network Training
• The Vision
Probabilistic Programming

ML: Algorithms & Applications

PL: Evaluators & Semantics

Stats: Inference & Theory

AI: Deep Learning
Existing Languages

Graphical Models
- BUGS
- STAN

Factor Graphs
- Factorie
- Infer.NET

Infinite Dimensional Parameter Space Models
- Anglican
- WebPPL

Unsupervised Deep Learning
- PYRO
- ProbTorch
Model-based reasoning and Inference
Radiographic Inspection for Oil Spill Prevention

Example - Tomography

Tom Rainforth

Automating Inference, Learning, and Design 8/22
In inference, we have

\[ p(X|Y) \propto p(X) \cdot p(Y|X) \]

where

\[ p(D_T|G, D_S) = N(D_T; \hat{D}P(G) + D_S, I) \]

\[ p(D_S|G) = N(D_S; \hat{D}S(G), I) \]

Radiographic Inspection for Oil Spill Prevention

[Rainforth, Mahendran, Osborne, Vedaldi and Wood. Industry sponsored project 2017]
Can you write a program to do this?

Mansinghka, Kulkarni, Perov, and Tenenbaum
Captcha Generative Model

(defm sample-char []
{:symbol (sample (uniform ascii))
 :x-pos (sample (uniform-cont 0.0 1.0))
 :y-pos (sample (uniform-cont 0.0 1.0))
 :size (sample (beta 1 2))
 :style (sample (uniform-dis styles))
 ...})

(defm sample-captcha []
(let [n-chars (sample (poisson 4))
  chars (repeatedly n-chars sample-char)
  noise (sample salt-pepper)
  ...]
  gen-image))
Conditioning

(defquery captcha [true-image]
  (let [gen-image (sample-captcha)]
    (observe (similarity-kernel gen-image)
             true-image)
    gen-image))

(doquery :ipmcmc captcha true-image)

Generative Model

Inference
2015: Probabilistic Programming

- Restricted (i.e. STAN, BUGS, infer.NET)
  - Easier inference problems -> fast
  - Impossible for users to denote some models
  - Fixed computation graph

- Unrestricted (i.e. Anglican, WebPPL)
  - Possible for users to denote all models
  - Harder inference problems -> slow
  - Dynamic computation graph

- Fixed, trusted model; one-shot inference
The AI/Repeated-Inference Challenge

“Bayesian inference is computationally expensive. Even approximate, sampling-based algorithms tend to take many iterations before they produce reasonable answers. In contrast, human recognition of words, objects, and scenes is extremely rapid, often taking only a few hundred milliseconds—only enough time for a single pass from perceptual evidence to deeper interpretation. Yet human perception and cognition are often well-described by probabilistic inference in complex models. How can we reconcile the speed of recognition with the expense of coherent probabilistic inference? How can we build systems, for applications like robotics and medical diagnosis, that exhibit similarly rapid performance at challenging inference tasks?”

Resulting Trend In Probabilistic Programming

- **Have fully-specified model?**
  - Yes
  - No

- **Inference?**
  - Yes
  - No

- **One-shot**
  - Probabilistic Programming

- **Repeated**
  - Inference Compilation
  - Unsupervised Deep Learning
Inference Compilation
Inference Compilation Desiderata

• Denote a model and inference problem as a probabilistic programming language program
• “Compile” for hours or days, depending on the problem, CPU/GPUs at disposal, etc.
• Get a “compilation artifact” controller that enables fast, repeated inference in the original model that is compatible with asymptotically exact inference
**Inference Compilation**

**Input**: an inference problem denoted in a probabilistic programming language

**Output**: a trained inference network (deep neural network “compilation artifact”)

---

Compiling Away Runtime Costs of Inference

Learn to invert the generative model, before seeing data

Objective function:

\[ J(\eta) = \int D_{KL}(\pi || q_{\lambda}) p(y) dy \]

\[ = \int p(y) \int p(x|y) \log \left( \frac{p(x|y)}{q(x|\varphi(\eta, y))} \right) dx dy \]

\[ = \mathbb{E}_{p(x,y)} \left[ -\log q(x|\varphi(\eta, y)) \right] + \text{const.} \]

Fully differentiable; can train entirely offline:

\[ \nabla_\eta J(\eta) = \mathbb{E}_{p(x,y)} \left[ -\nabla_\eta \log q(x|\varphi(\eta, y)) \right] \]
Example: Non-Conjugate Regression Graphical Model

**Finite Graphical Model**

\[ w_d \sim \text{Laplace}(0, 10^{1-d}) \quad \text{for } d = 0, 1, 2; \]
\[ t_n \sim t_r(w_0 + w_1 z_n + w_2 z_n^2, \epsilon^2) \quad \text{for } n = 1, \ldots, N \]
\[ \nu = 4, \epsilon = 1, \text{and } z_n \in (-10, 10) \]

**Inverted Graphical Model**

**Neural net proposal**

\[ q(w_{0:2}|z_{1:N}, t_{1:N}) \]

- Two layer MLP
- 200 units
- 3-component MOG for each output
Figure 1: Representative output in the polynomial regression example. Plots show 100 samples each at 5% opacity, with the mean marked as a solid dashed line. These are all proposed using the same neural network — not just the same neural network structure, but also identical learned weights. The MCMC posterior is generated by thinning 10000 samples by a factor 100, after 10000 samples of burnin. The neural network proposal density for the weights yields estimated polynomial curves very close to the true posterior solution, albeit slightly more diffuse. Any small mismatch is easily corrected via importance reweighing.

structure are shown in Figure 2. Here we place a Laplace prior on the regression weights, and have Student-t likelihoods, giving us $w_d \sim \text{Laplace}(0, 10^{-1})$ for $d = 0, 1, 2$; $t_n \sim \text{t}_4(w_0 + w_1 z_n + w_2 z_n^2, 1)$ for $n = 1, \ldots, N$ for fixed $\nu = 4$, $\nu = 1$, and we place a uniform prior on $(10, 10)$ for $z_n$. The goal is to estimate the posterior distribution of weights for the constant, linear, and quadratic terms, given any possible collected dataset $\{z_n, t_n\}_{n=1}^N$. In the notation of the surrounding sections, we have latent variables $x \sim \{w_0, w_1, w_2\}$ and observed variables $y \sim \{z_n, t_n\}_{n=1}^N$.
CSIS: Inf. Comp. for Higher-Order PPL

- Same IS proposal learning objective

\[ \nabla_\eta \mathcal{J}(\eta) = \mathbb{E}_{p(x,y)} \left[ -\nabla_\eta \log q(x|\varphi(\eta, y)) \right] \]

- Inf. dim. graph so only evaluation/“forward” inference methods possible, i.e. SIS with unnormalized weights

\[
\frac{\prod_{i=1}^{N} g_i(y_i|\phi_i) \prod_{j=1}^{M} f_j(x_j|\theta_j)}{\prod_{j=1}^{M} q(x_j|\varphi(\eta, y, x_{1:j-1}))} = w^{(k)}
\]

- “Forward”-structured proposal / controller

\[
q(x|\varphi(\eta, y)) \triangleq \prod_{j=1}^{M} q(x_j|\varphi(\eta, y, x_{1:j-1}))
\]
A. Generating synthetic training data

Generating synthetic training data involves creating data that mimics real-world data but is not derived from real data. In the context of the problem at hand, this means creating Captcha images that are similar to those encountered in real-world scenarios but are generated synthetically. This is crucial for training models to be robust against various types of Captchas.

The process involves the following steps:

1. **Sampling**: Taking random samples from the real data distribution.
2. **Generating**: Using these samples to generate new data points that are similar to the real data but not identical.
3. **Training**: Using these synthetic data points to train the model.

The key here is to ensure that the synthetic data is diverse and covers all possible variations that might be seen in real-world data. This helps the model generalize better and perform well on unseen data.

In particular, the style shown in Table I, we use different settings of the prior variables including the character identities. For each different generating synthetic data generator corresponds to the model synthetic data generation effort. The corresponding per-style renderer and its fidelity being the primary component of the Captcha renderer letter identities. Given these, we use a custom stochastic as kerning and various style-specific deformations, and parameter set controlling Captcha-rendering parameters such latent structured random variable mind that there is a separate unique model for each style. The following equations we omit the style subscript while keeping in Wikipedia, Facebook) involving distinct character ranges, how to generate both the latent random variable joint densities.

\[
\mathbf{x}^{(n)} = \left[ x_1^{(n)} \, x_2^{(n)} \, \cdots \, x_t^{(n)} \, \cdots \right]
\]

\[
\mathbf{y}^{(n)}
\]

\[
x^{(n)}, y^{(n)} \sim p(x, y)
\]

\[
q = \]

\[
\begin{align*}
\text{Sampling} & \quad \sim \\
\text{Output} & \\
\text{LSTM} & \\
\text{Input} & \\
\text{CNN} & \\
\end{align*}
\]

## Captcha Breaking

<table>
<thead>
<tr>
<th>Type</th>
<th>Baidu (2011)</th>
<th>Baidu (2013)</th>
<th>eBay</th>
<th>Yahoo</th>
<th>reCaptcha</th>
<th>Wikipedia</th>
<th>Facebook</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our method</td>
<td>RR 99.8%</td>
<td>99.9%</td>
<td>99.2%</td>
<td>98.4%</td>
<td>96.4%</td>
<td>93.6%</td>
<td>91.0%</td>
</tr>
<tr>
<td></td>
<td>BT 72 ms</td>
<td>67 ms</td>
<td>122 ms</td>
<td>106 ms</td>
<td>78 ms</td>
<td>90 ms</td>
<td>90 ms</td>
</tr>
<tr>
<td>Bursztein et al. [15]</td>
<td>RR 38.68%</td>
<td>55.22%</td>
<td>51.39%</td>
<td>5.33%</td>
<td>22.67%</td>
<td>28.29%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT 3.94 s</td>
<td>1.9 s</td>
<td>2.31 s</td>
<td>7.95 s</td>
<td>4.59 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starostenko et al. [16]</td>
<td>RR</td>
<td></td>
<td></td>
<td>91.5%</td>
<td>54.6%</td>
<td>&lt; 0.5 s</td>
<td></td>
</tr>
<tr>
<td>Gao et al. [18]</td>
<td>RR 34%</td>
<td></td>
<td></td>
<td>55%</td>
<td>34%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td>51%</td>
<td></td>
<td>36%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goodfellow et al. [6]</td>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td>99.8%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stark et al. [8]</td>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>90%</td>
<td></td>
</tr>
</tbody>
</table>

### Facebook Captcha

<table>
<thead>
<tr>
<th>Observed images</th>
<th>W4kgvQ (W4kgvQ)</th>
<th>uV7FeWB</th>
<th>Mqhnpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference</td>
<td>10^7 W4kgvQ</td>
<td>uV7FeWB</td>
<td>Mqhnpt</td>
</tr>
<tr>
<td>Training traces</td>
<td>10^6 WA4rjvQ</td>
<td>uV7FeWB</td>
<td>MypppT</td>
</tr>
<tr>
<td></td>
<td>10^5 Woxewd9</td>
<td>mTTEMm</td>
<td>RIpES</td>
</tr>
<tr>
<td></td>
<td>10^4 BKvu2Q</td>
<td>C9QDsoN</td>
<td>rS5FP2B</td>
</tr>
</tbody>
</table>

Le TA, Baydin AG, Zinkov R, **Wood F.** Using Synthetic Data to Train Neural Networks is Model-Based Reasoning, IJCNN. 2017.
The Quest for New Physics
Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model. Submitted to NIPS.
High Peaks -- Our Battle to Control a SHERPA

- Controlled by PyProb
  - C++ prior model
  - SHERPA; 1M+ lines
  - Describes standard model
  - Only interface via intercepted U(0,1) RV's
  - Python likelihood
    - ATLAS detector component simulator
- Inf. comp. artifact first ever inference using LHC generative model software stack
- Neural network SHERPA controller 1000x's more efficient than MH or IS inference; potential for real-time

![Feynman diagrams](image1)

**Figure 1:** Top: branching ratios of the tau lepton, effectively the prior distribution of the decay channel in the SHERPA simulation. Note that the scale is logarithmic. Bottom: Feynman diagrams for tau decays illustrating those that can produce multiple detected particles.

![Corner plot](image2)

**Figure 2:** The corner plot on the left shows the particle energies of the two most energetic final state particles and their joint probability. To the right, the distribution of the originating momentum components of the \( \tau \) lepton and its decay mode is shown. In the middle we show the event composition as characterized by the number of primarily electromagnetically interacting and hadronically interacting final state particles as well as the number of decay products. To the right we show the original observation as well as the mean observation generated during inference.

![Bottom-up model](image3)

**Bottom-up model**

Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model. Submitted to NIPS.
Key Benefit

- Instant interpretability

---

Figure 1: An illustration of the framework, where the pyprob package in Python is controlling SHERPA in C++ through the probabilistic programming execution (PPX) protocol. The bottom part shows the actual probabilistic structure of the $\tau$ decay simulator, excluding uncontrolled addresses. Inference algorithms to provide interpretable posterior samples. In this work, we take this approach, extend previous work in universal probabilistic programming [15] and inference compilation [16] to large-scale complex simulators, and demonstrate the ability to execute existing simulator codes under the control of general-purpose inference engines. This is achieved by a cross-platform probabilistic execution protocol (Figure 1) through which an inference engine can control simulators in a language-agnostic way. We implement a range of general-purpose inference engines from the Markov chain Monte Carlo (MCMC) [17] and importance sampling [18] families. The execution framework we develop currently has bindings in C++ and Python, which are languages of choice for many large-scale projects in science and industry, and it can be used by any other language pending the implementation of a lightweight front end.

We demonstrate the technique in a particle physics setting, introducing probabilistic programming as a novel tool to determine the properties of particles at the Large Hadron Collider (LHC) at CERN. This is achieved by coupling our framework with SHERPA [21], a state-of-the-art Monte Carlo event generator of high-energy reactions of particles, which is commonly used with GEANT [22], a toolkit for the simulation of the passage of the resulting particles through detectors. In particular, we perform inference in the case of $\tau$ (tau) lepton particle decay in a realistic detector, controlling the simulation within the standard SHERPA software with minimal modification and extracting posterior distributions in agreement with MCMC-based ground truths. To our knowledge this is the first time that universal probabilistic programming has been applied in this domain and in this scale, controlling a codebase of nearly 1M lines of code. Our approach is readily scalable to more complex events and full detector simulators, paving the way to its use in the discovery of new fundamental physics.

---

1 Simulation of High-Energy Reactions of Particles. https://sherpa.hepforge.org/
Code to Try

**PySPPL Compiler**
https://github.com/Tobias-Kohn/PyPPLCompiler
9692 lines of Python
Compiles a first-order subset of Python to a graphical model

**Pyfo**
https://github.com/brodyrant/pyfo
14193 lines of Python
Pure Python STAN replacement + discrete RVs

**PyProb**
https://github.com/probprog/pyprob.git
4958 lines of Python
Pure Python PPL + Inference Compilation

**FOPPLCompiler**
git@github.com:probprog/foppl-compiler
Graphical-model inversion

**Anglican-lite**
https://bitbucket.org/brx/anglican-lite
Clojure FOPPL->graphical model compiler

**Anglican**
http://anglican.ml
Clojure HOPPL

goo.gl/iogdVz

https://bitbucket.org/probprog/ppaml-summer-school-2016
Challenges

• Sharply peaked likelihoods
  • Adversarial training?
• Efficient forward model execution at test time
  • Surrogates?
• Deep neural network training at scale
Massively Distributed Deep Network Training
Key Ideas

• Asynchronous distributed SGD (aka Hogwild) does not work “at scale”
• Chen et al (Bengio/Google) [2016] suggest “obvious” idea
  • Drop straggling workers in synchronous SGD
  • Provided mini-batch data selection is uncorrelated with worker identity
    this is completely kosher in expectation

• Our idea
  • Learn a deep nonlinear dynamical system model of cluster performance
    and use order statistics from said model to drop straggling workers

• TL&DR
  • Higher throughput leads to faster training times despite dropping
    gradient mini-batch computations
  • Learned model does better than simple heuristics
The first model of runtimes we consider assumes that they are independent and identically distributed (iid) Gaussian. Under the assumption that $x_j = N(\mu_x, \sigma_x^2)$, the distribution of each order statistic $p(\tilde{x}(1)), p(\tilde{x}(2)), \ldots, p(\tilde{x}(n))$ is independent and $E[\tilde{x}(1)] \leq E[\tilde{x}(2)] \leq \cdots \leq E[\tilde{x}(n)]$.

Under the given iid normality assumption the distribution of each order statistic has a closed form:

$$p(\tilde{x}(j)) = \frac{Z(n, j)}{Z(1)} x\left[\left(\frac{x}{\mu_t} + \frac{1}{\sqrt{n}}\right)^2\right]^{j-1} \left(\frac{x}{\mu_t} - \frac{1}{\sqrt{n}}\right)^{n-j} \frac{1}{\left[1 - \left(\frac{x}{\mu_t} + \frac{1}{\sqrt{n}}\right)^2\right]^{n/2}}$$

where $x$ is the cumulative distribution function (CDF) of $N(\mu_t, \sigma_t^2)$ and $Z(n, j) = \frac{n!}{(j-1)! (n-j)!}$.

Note that each order statistic's distribution, including the maximum, increases as the variance of the runtime distribution increases, while the average runtime does not.

As a baseline in subsequent sections we will use a useful approximation of the expectations of order statistics under this iid normality assumption. This is known as the Elfving (1947) formula (Royston, 1982):

$$E[\tilde{x}(j)] \approx \mu_t + \frac{1}{\sqrt{n}}\left(\frac{1}{j} - \frac{1}{n+1}\right)$$

In practice, the parameters $\mu_t, \sigma_t$ are fit using maximum likelihood on the first fixed lagged window. Here, we note that the Elfving model requires full observability of runtimes to dynamically compute this. Further, it is not known how to derive the analytic form of the joint order statistic distribution of non-Gaussian distributed correlated random variables. However, a Monte Carlo approximation of the order statistics is straightforward: use a model to predict the joint distribution of the $x_j$'s, then sample, sort, and record the values of all $n$ sorted samples, and then repeat. Towards that end we developed a model of correlated compute times from which we will then be able to construct Monte Carlo order statistic estimates for use in determining the optimal cutoff threshold.

Figure 1: Results of throughputs given by amortized inference. Each runtime plot (5 surrounding the top figure) shows the individual runtimes of the worker (x-axis index) during an iteration of SGD on a 158 node cluster. We highlight SGD iterations 1, 50, 100, 150, and 200 which highlight two significantly different regimes of persistent time-and-machine-identity correlated worker runtimes. The top large figure displays a comparison of throughputs achieved by waiting for all workers to finish (green) and using the inferred cutoff method (red) relative to the ground truth maximum achievable (oracle). The bottom figure displays the reduction in time per iteration when Cutoff SGD is used.
Figure 2: Runtime profiles of various iterations of SGD of the validation set in our training step. The maximum throughput cutoff under the model predictions is shown in red, indicating a large chunk of idle time is reduced as a result of stopping early. (A/B): selected observed runtimes vs predicted runtime order statistics for a 158 node cluster. Notably, when there are exceptionally slow workers present, the cutoff is set to proceed without any of them as seen in figure (A). (C/D): example predicted vs. actual runtimes for 2175 node cluster. All predicted order statistics are shown with ±2 standard deviations.
Improved Training Time on MNIST

Figure 3: MNIST validation loss convergence for our model based methods, Elfving and Bayesian, and popular approaches. Batch size - 10112, learning rate scaled to 0.64 for sync and 0.004 (0.64 / num workers) for async.
Improved Training Times for Large Neural Networks

Figure 4: We trained ResNet-64 and the WideResNet with 28 Layers and a width factor of 10. Both networks are trained on CIFAR-100, with a batch-size of 47850 on ResNet-64 and 8700 on WideResNet. All training curves train to 1100 iterations. Initial learning rates for both are set to 0.16, with a 20% decay on WideResNet at $t=500$ and $t=1000$. The plots show in order: total throughput over training, wall-clock validation accuracy over training time, and the ratio of training time vs. inter-rank communication cost. On the validation loss curve in the center, dashed vertical lines indicate when the final iteration completed.
Vision

• A learned computational artifact for rapid, even real-time, interpretable LHC event processing
  • Trigger
• A framework for model criticism and new physics using high-quality importance sampling-based evidence estimates
• A framework for efficiently training simulator controllers for various industry applications
  • Leverage existing simulator code
  • Use general purpose compute
  • Useful for realtime anomaly detection, advanced analytics, etc.
Thank You

- People: **Gunes Baydin**, Wahid Bhimji, Lukas Heinrich, Kyle Cranmer, Tuan Anh Le, Jan Willem van de Meent, Hongseok Yang, Brooks Paige, David Tolpin, amongst many others

- Funding: Intel, DARPA, NSERC
Online Learning Rate Adaptation with Hypergradient Descent

Atılım Güneş Baydin\textsuperscript{1}  Robert Cornish\textsuperscript{1}  David Martínez Rubio\textsuperscript{2}

Mark Schmidt\textsuperscript{3}  Frank Wood\textsuperscript{1}

\textsuperscript{1}Department of Engineering Science, University of Oxford, Oxford, United Kingdom
\textsuperscript{2}Wadham College, University of Oxford, Oxford, United Kingdom
\textsuperscript{3}Department of Computer Science, University of British Columbia, Vancouver, BC, Canada
\
\{gunes,rcornish,fwood\}@robots.ox.ac.uk
\texttt{david.martinez2@wadh.ox.ac.uk}  \texttt{schmidtm@cs.ubc.ca}

Abstract

We introduce a general method for improving the convergence rate of gradient-based optimizers that is easy to implement and works well in practice. We analyze the effectiveness of the method by applying it to stochastic gradient descent, stochastic gradient descent with Nesterov momentum, and Adam, showing that it improves upon these commonly used algorithms on a range of optimization problems; in particular the kinds of objective functions that arise frequently in deep neural network training. Our method works by dynamically updating the learning rate during optimization using the gradient with respect to the learning rate of the update rule itself. Computing this "hypergradient" needs little additional computation, requires only one extra copy of the original gradient to be stored in memory, and relies upon nothing more than what is provided by reverse-mode automatic differentiation.

1 Introduction

In nearly all gradient descent algorithms the choice of learning rate remains central to efficiency; Bengio \cite{4} asserts that it is "often the single most important hyper-parameter" and that it always should be tuned. This is because choosing to follow your gradient signal by something other than the right amount, either too much or too little, can be very costly in terms of how fast the overall descent procedure achieves a particular level of objective value.

Understanding that adapting the learning rate is a good thing to do, particularly on a per parameter basis dynamically, led to the development of a family of widely-used optimizers including AdaGrad \cite{9}, RMSProp \cite{28}, and Adam \cite{14}. However, a persisting commonality of these methods is that they are parameterized by a "pesky" fixed global learning rate hyperparameter which still needs tuning.

There have been methods proposed that do away with needing to tune such hyperparameters altogether \cite{21} but their adoption has not been widespread, owing perhaps to their complexity, applicability in practice, or performance relative to the aforementioned family of algorithms.

Our initial conceptualization of the learning rate adaptation problem was one of automatic differentiation \cite{2}. We hypothesized that the derivative of a parameter update procedure with respect to its global learning rate ought to be useful for improving optimizer performance. This conceptualization is not unique, having been explored, for instance, by Maclaurin et al. \cite{16}. While the automatic differentiation perspective was integral to our conceptualization, the resulting algorithm turns out to simplify elegantly and not require additional automatic differentiation machinery. In fact, it is easily adaptable to nearly any gradient update procedure while only requiring one extra copy of a gradient to be held in memory and very little computational overhead; just a dot product in the dimension of the parameter. Considering the general applicability of this method and adopting the

Simple Idea

- Dynamically adjust the learning rate in gradient descent by using automatic differentiation to differentiate wrt the learning rate through the gradient update procedure itself

\[ \theta_t = \theta_{t-1} - \alpha \nabla f(\theta_{t-1}) \]

\[ \theta_{t-1} = \theta_{t-2} - \alpha \nabla f(\theta_{t-2}) \]

Noting that the chain rule yields

\[ \frac{\partial f(\theta_{t-1})}{\partial \alpha} = \nabla f(\theta_{t-1}) \cdot \frac{\partial(\theta_{t-2} - \alpha \nabla f(\theta_{t-2}))}{\partial \alpha} = \nabla f(\theta_{t-1}) \cdot (-\nabla f(\theta_{t-2})) \]

suggesting a simple learning rate update rule

\[ \alpha_{t} = \alpha_{t-1} - \beta \frac{\partial f(\theta_{t-1})}{\partial \alpha} = \alpha_{t-1} - \beta \nabla f(\theta_{t-1}) \cdot \nabla f(\theta_{t-2}) \]

Logistic Regression (MNIST) MLP (MNIST) VGG (CIFAR-10)
Extras
Model-Based Reasoning
Perception / Inverse Graphics

Captcha Solving

\[ y \]  Input Image
\[ \text{Intermediate Iterations} \]
\[ x \]  Final Inferred Image

Scene Description

\[ y \]  Observed Image
\[ x \]  Inferred (reconstruction)
\[ \text{Inferred model re-rendered with novel poses} \]
\[ \text{Inferred model re-rendered with novel lighting} \]

\[ x \]
scene description

\[ y \]
image


Reasoning about reasoning

Want to meet up but phones are dead...

I prefer the pub.
Where will Noah go?
Simulate Noah:
Noah prefers pub
but will go wherever Andreas is
Simulate Noah simulating Andreas:
...
-> both go to pub

Stuhlmüller, and Goodman.
"Reasoning about reasoning by nested conditioning: Modeling theory of mind with probabilistic programs."
Directed Procedural Graphics

Stable Static Structures

Figure 1: Physical realizations of stable structures generated by our system. To create these structures, we write programs that can synthesize examples that their users might never have thought of independently.

Figure 2: Example designs of complex structures. a) Six protruding ledges; on the right, it grows through and around them. b) A crescent-like shape. c) Gnarly trees grow around obstacles.

Figure 3: SOSMC sampling from a random building complex and designing stable stacking structures. We implement HMC in a high-performance probabilistic programming language, and we evaluate its ability to generate random structures (e.g. a random tower or a randomly-perturbed arch), constrain the output of the program to be near static equilibrium, and then sample from the constrained output space using Hamiltonian Monte Carlo.

Directed Procedural Graphics

Procedural Graphics


\[ x \rightarrow \text{simulation} \rightarrow y \rightarrow \text{constraint} \]
Program Induction

\( \tilde{y} \sim p(\cdot | x) \)

\( x \sim p(x) \)

\( \tilde{y} \sim p(\cdot | x) \)

\( x \sim p(x \mid y) \)

Program source code

```
(lambda (stack-id) (safe-uc (* (if (< 0.0 (* (+ -1.0 (begin (define _1147 (safe-uc 1.0 1.0) 0.0)) (* 0.0 (+ 0.0 (safe-uc (* (* (dec -2 .0) (safe-sqrt (begin (define _1140 (3.14159) (safe-log -1.0)))) 2.0) 0.8)))) 1.0)) (+ (safe-div (begin (define _1149 (* (+ 3.14159 -1.0) 1.0)) 1.0) 0.0) (safe-log 1.0)) (safe-log -1.0)) (begin (define _11 ...)
```

Program output

Perov and Wood.

"Automatic Sampler Discovery via Probabilistic Programming and Approximate Bayesian Computation"

AGI (2016).
Reinforcement Learning


# PPAML Week-Long Summer School: 1.5 Days of Student Coding

**Day 1**
- **9:00 – 10:00** Intro to Summer School (consent forms, etc.)
- **10:00 – 10:30** Galois: Overview of PF
- **10:30 – 12:00** Hands-On: Functional Programming (Galois)
- **12:00 – 1:30** Lunch
- **1:30 – 2:30** Lecture: Intro to Prob. Prog. (Invrea - Wood)
- **2:30 – 3:00** Lecture: Intro to Inference (Invrea - Paige)
- **3:00 – 5:00** Hands-On: Probabilistic & Generative Modeling

**Day 2**
- **9:00 – 10:00** Lecture Intro to Functional Programming and Clojure
- **10:00 – 10:30** Hands-On: Functional Programming
- **10:30 – 12:00** Hands-On: Anglican Programming
- **12:00 – 1:30** Lunch
- **1:30 – 2:30** Lecture: Contributing to Anglican (Invrea - van de Wetering)
- **2:30 – 3:00** Lecture: Introduction to Anglican Programming
- **3:00 – 5:00** Hands-On: Project Free Coding

**Day 3**
- **9:00 – 10:00** Hands-On: Project Free Coding
- **10:00 – 10:30** Hands-On: Anglican Programming
- **10:30 – 12:00** Hands-On: Project Free Coding
- **12:00 – 1:30** Lunch
- **1:30 – 2:30** Lecture: Advanced Prob. Prog. (Invrea - Paige)
- **2:30 – 3:00** Hands-On: Project Free Coding
- **3:00 – 5:00** Project Presentations
Scene Generation

Generative Model for Images

Sample Facts → Sample sprites + positions constrained by facts → Render Image

{sprite :boy x 100 y 100 :faces :left}

{sprite :bear x 10 y 100 :faces :right}

Generative Model for Captions

Facts → Select words and pronouns → Generate Sentence

[faces :bear :girl], [:close :boy :bear]

"The girl is faced by the bear, which is close to the boy."

Conditioning

- Condition image on sentence
  "The ball kicks Bob while the bear is faced by Alice."

Image

Facts

Sentence

x

facts

y

images and sentences

Alex Collins (acollins@nvidia.com)
Alex Ledger (a.led1027@gmail.com)
Timon Gehr (timon.gehr@gmail.com)
Problem: Find the location of objects/regions with an unknown appearance.

“Anglican = awesome”

“Spent two weeks trying to get the model working with Figaro / Scala”

“It took me 1 evening (at the bar with cocktails) to make it work with Anglican / Clojure”

\[ x \] unknown locations
\[ y \] fixations
Rock Composition Via X-ray Fluorescence

Matthew Dirks
mcdirks@cs.ubc.ca

x

rock composition

y

x-ray fluorescence
Thinking Generatively about Discriminative Tasks

(defquery lin-reg [x-vals y-vals]
(let [m (sample (normal 0 1))
     c (sample (normal 0 1))
     f (fn [x] (+ (* m x) c))]
  (map (fn [x y]
         (observe
          (normal (f x) 0.1) y))
       x-vals y-vals))
  [m c])

(doquery :ipmcmc lin-reg data options)

([[0.58 -0.05] [0.49 0.1] [0.55 0.05] [0.53 0.04] ....]
(defquery md5-inverse [L md5str]
 "conditional distribution of strings
 that map to the same MD5 hashed string"
 (let [mesg (sample (string-generative-model L))]
   (observe (dirac md5str) (md5 mesg)
     mesg))))
Decision-Making Under Uncertainty

INVREA
Make Better Decisions

£23B/yr risk analytics market

ORACLE
Crystal Ball

spreadsheet model

actuals

https://invrea.com/plugin/excel/v1/download/
Why Model-Based Reasoning?

- Interpretability
- Source of labeled data
- Regularization
  - Computation structure
- Domain knowledge
Inference Compilation
First: Graphical Model Inference

**Goal:** efficient posterior inference in generative models with latent variables \( x \) and observed variables \( y \)

\[
p(x, y) \triangleq \prod_{i=1}^{N} p(x_i | \text{PA}(x_i)) \prod_{j=1}^{M} p(y_j | \text{PA}(y_j))
\]
Inference

Goal: efficient posterior inference in generative models with latent variables $x$ and observed variables $y$

$$p(x, y) \triangleq \prod_{i=1}^{N} p(x_i | PA(x_i)) \prod_{j=1}^{M} p(y_j | PA(y_j))$$

e.g. importance sampling and SMC approximate the posterior $\pi(x) \equiv p(x | y)$ as weighted samples

$$\hat{p}(x | y) = \sum_{k=1}^{K} W_k \delta_{x_k}(x) \quad w(x) = \frac{p(x, y)}{q(x | \lambda)} \quad W_k = \frac{w(x_k)}{\sum_{j=1}^{K} w(x_j)}$$

Performance depends on quality of proposal $q(x | \lambda)$
One Notion of Optimal Proposal

Learning an importance sampling proposal for a single dataset

Target density $\pi(x) = p(x|y)$, approximating family $q(x|\lambda)$

Single dataset $y$: $\arg\min_{\lambda} D_{KL}(\pi||q_{\lambda})$

$D_{KL}[q_{\phi}(x|y)||p_{\theta}(x|y)]$

Figure 1: The Gaussian $q$ which minimizes $\alpha$-divergence to $p$ (a mixture of two Gaussians), for varying $\alpha$. $\alpha \to -\infty$ prefers matching one mode, while $\alpha \to \infty$ prefers covering the entire distribution.

Open-Universe Gaussian Mixture Model

1: procedure GMM
2: \( K \sim p(K|\cdot) \)
3: for \( k = 1, \ldots, K \) do
4: \( \mu_k, \Sigma_k \sim p(\mu_k, \Sigma_k|\cdot) \)
5: \( \pi \leftarrow \text{uniform}(1, K) \)
6: for \( n = 1, \ldots, N \) do
7: \( z_n \sim p(z_n|\pi) \)
8: \( y_n \sim p(y_n|z_n = k, \mu_k, \Sigma_k) \)
return \( \{\mu_k, \Sigma_k\}_{k=1}^{K}, K \)

▷ sample number of clusters
▷ sample cluster parameters
▷ sample class label
▷ sample or observe data
GMM Inference

Kernel density estimation of the distribution over maximum a-posteriori values of the means \( \{\max_{\mu_k} p(\mu_k|y)\}^3_{k=1} \) over 50 independent runs

**Top:** SMC  
**Bottom:** CSIS
Effect of Training Duration

Observed points

Training traces

$10^7$

$10^4$
Synthetic Data for Training Deep Networks

Goodfellow, Bulatov, Ibarz, Arnoud, Shet; Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. 2014.

Jaderberg, Simonyan, Vedaldi, Zisserman; Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition. 2014.

Gupta, Vedaldi, Zisserman; Synthetic Data for Text Localisation in Natural Images. 2016.

Le, Baydin, Zinkov & Wood. Using Synthetic Data to Train Neural Networks is Model-Based Reasoning IJCNN 2017.
Advanced Topics Take-Homes

• If you have an existing simulator it is, in principle, possible to perform inference in it now (without re-coding it), using it as a prior in a Bayesian sense

• Amortized inference is powerful and works for the same reason that deep neural networks trained on synthetic data work
Challenges

• Probabilistic Programming
  • Nesting/compositionality
  • Automatically factoring inference
  • Finite approximations

• Inference Compilation
  • Inference network structure from generative model or vice versa
  • Forward model surrogates

• Model Learning
  • PLs that support
    • AESMC-style ELBOs
  • Model learning through discrete random primitives
Recent Papers

Published


Submitted

1. Revisiting Reweighted Wake-Sleep. Submitted to *NIPS*; on *arXiv*
2. **Faithful Inversion of Generative Models for Effective Amortized Inference. Submitted to *NIPS*; on *arXiv**
3. Bayesian Distributed Stochastic Gradient Descent. Submitted to *NIPS*
4. Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model. Submitted to *NIPS*.
5. Hamiltonian Monte Carlo for Probabilistic Programs with Discontinuities. Submitted to *NIPS*; on *arXiv*
6. **Inference Trees: Adaptive Inference with Exploration. Submitted to *NIPS.**

Ms Theses

1. W. Harvey. Automatic Ingestion of Plot Data, MS Thesis, Oxford, 2018
TL&DR

- Programming languages can be used to denote inference problems.
- There are at least two families of probabilistic programming languages; one can be compiled to graphical models or factor graphs, the other, corresponding in character to normal, everyday programming languages, cannot.
- It is possible to develop generic and reasonably efficient inference algorithms for both families.
- There is a rapidly emerging connection between probabilistic programming, variational inference, and differential programming that could give rise to the next generation of AI tools.
- There are all kinds of interesting research and engineering challenges remaining.