Inference Compilation

or How I Learned to Stop Worrying and Love Deep Networks

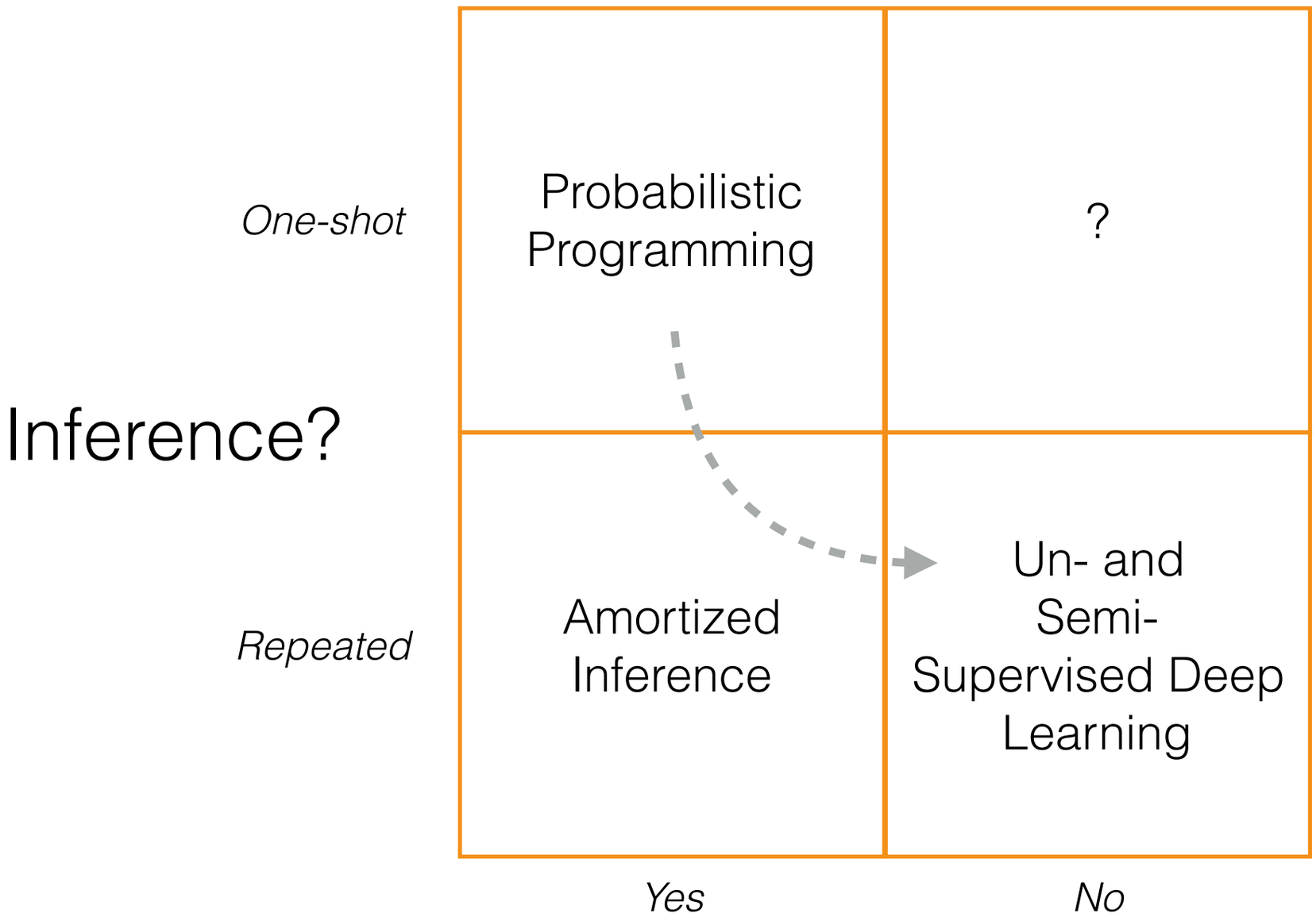
Frank Wood
fwood@cs.ubc.ca

2014/5 : 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, Venture)
  - Possible for users to denote all models
  - Harder inference problems -> slow
  - Dynamic computation graph

- Fixed, trusted model; one-shot inference

Trend In Probabilistic Programming



Have fully-specified model?

Proposal : Zoubin diagram?

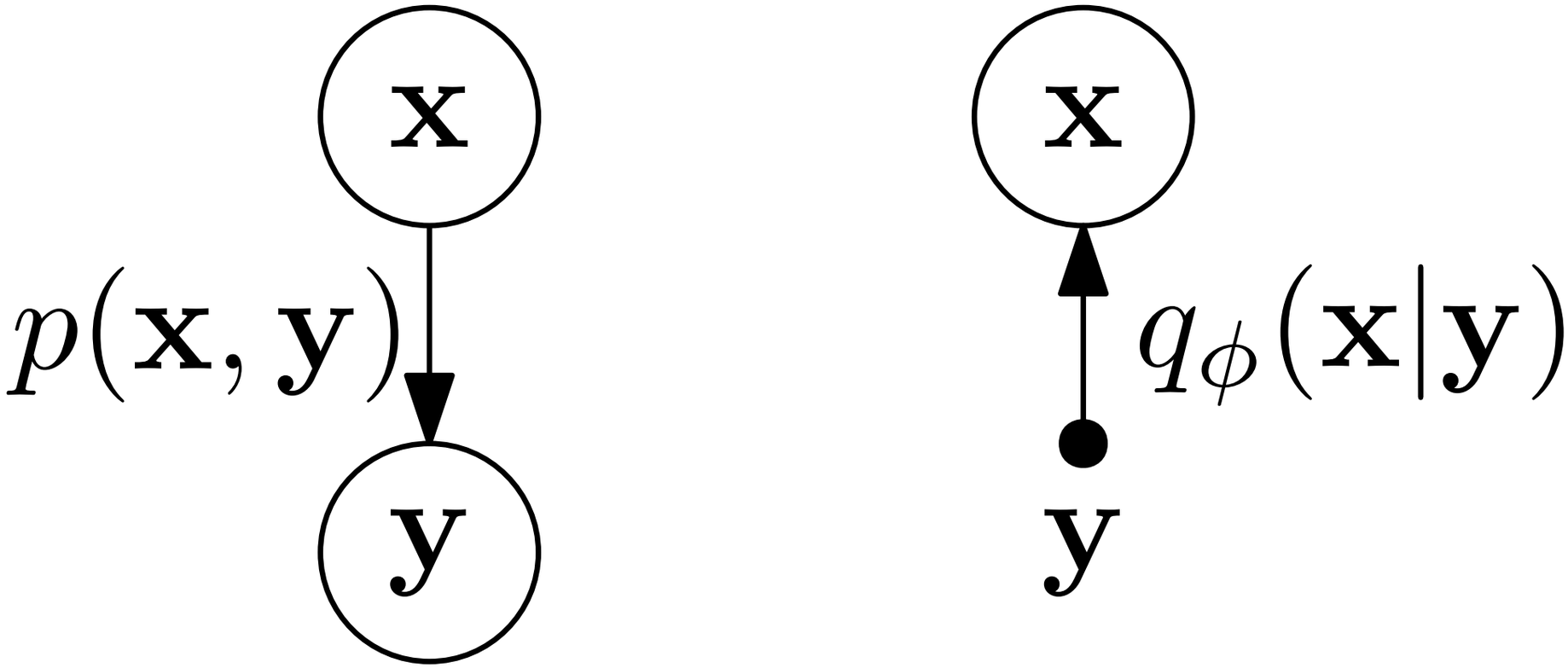
Problems

1. Bayesian inference is computationally expensive
   - Amortized inference
     - Background
     - Types
     - Examples
       - First-order
       - Higher-order
2. Determining neural-net structure is hard
   - Automated inference neural network stub generation
3. Writing models is hard
   - Simultaneous model-learning
     - Revisit wake-sleep?

"**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?"

Stuhlmüller A, Taylor J, Goodman N. Learning stochastic inverses. In Advances in Neural Information Processing Systems 2013 (pp. 3048-3056).

Amortized Inference



Related Ideas

- Hinton, Dayan, Frey, and Neal. The "**wakesleep**" algorithm for unsupervised neural networks. Science, 1995.

- *Importance sampling (optimal proposal)*
  - Ortiz and Kaelbling. Adaptive importance sampling for estimation in structured domains. UAI, 2000.
  - Cornebise, Moulines, and Olsson. Adaptive methods for sequential importance sampling with application to state space models. Statistics and Computing, 2008.
  - Gu, Ghahramani, and Turner. Neural adaptive sequential Monte Carlo. NIPS, 2015.
- *Variational inference*
  - Kingma and Welling. Auto-Encoding Variational Bayes. ICLR 2014
  - Kulkarni, Whitney, Kohli, Tenenbaum. Deep convolutional inverse graphics network. NIPS, 2015.
- *Other*
  - Shotton et al. Real-Time Human Pose Recognition in Parts from Single Depth Images. CVPR 2011
  - Goodfellow, Bulatov, Ibarz, Arnoud, Shet; Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. 2014.
  - Kulkarni, Kohli, Tenenbaum, Mansinghka. Picture: A Probabilistic Programming Language for Scene Perception. CVPR, 2015

“Guide Program”

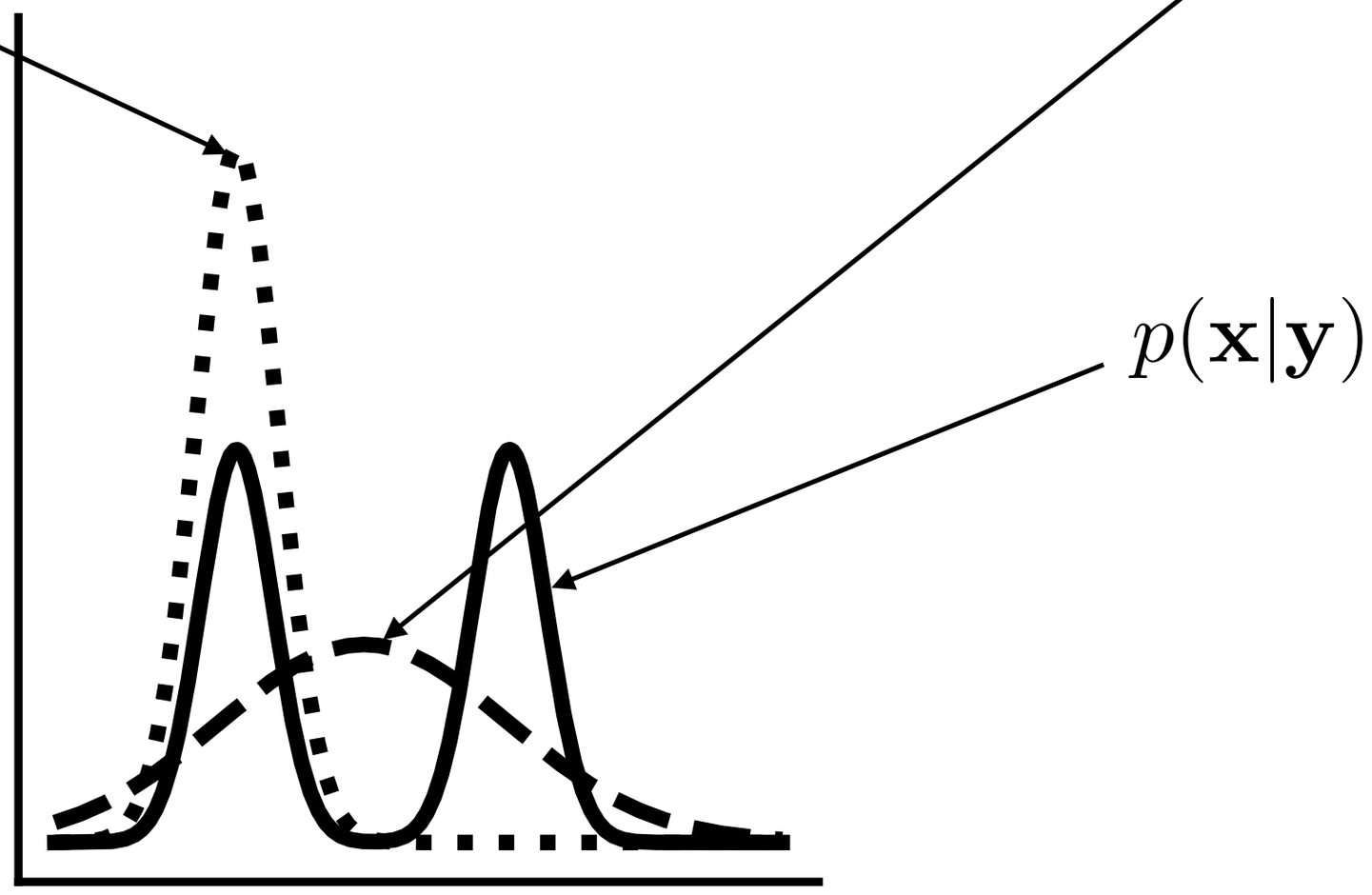
$$\underset{q_\phi}{\operatorname{argmin}}\, \mathrm{KL}(q_\phi(\mathbf{x}|\mathbf{y})||p(\mathbf{x}|\mathbf{y}))$$

$$\sum_i \left[\mathbb{E}_{q_\varphi(\mathbf{x}|\mathbf{y}_i)} [\log \frac{p_\theta(\mathbf{x}, \mathbf{y}_i)}{q_\varphi(\mathbf{x}|\mathbf{y}_i)}] \right]$$

“Inference Network”

$$\underset{q_\phi}{\operatorname{argmin}}\, \mathrm{KL}(p(\mathbf{x}|\mathbf{y})||q_\phi(\mathbf{x}|\mathbf{y}))$$

$$\mathbb{E}_{p_\theta(\mathbf{y})} \left[\mathbb{E}_{p_\theta(\mathbf{x}|\mathbf{y})} [\log \frac{p_\theta(\mathbf{x}|\mathbf{y})}{q_\varphi(\mathbf{x}|\mathbf{y})}] \right]$$



http://www.tuananhle.co.uk/notes/amortized-inference.html

Minka T. Divergence measures and message passing. Tech report. 2005.

Training an “Inference Network”

$$\mathcal{L}(\phi) = \mathbb{E}_{p(\mathbf{y})}\left[\mathrm{KL}(q_\phi(\mathbf{x}|\mathbf{y})||p(\mathbf{x}|\mathbf{y})\right]$$

$$= \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\left[-\log q_\phi(\mathbf{x}|\mathbf{y})\right] + \text{const.}$$

Minimize using stochastic gradient descent:

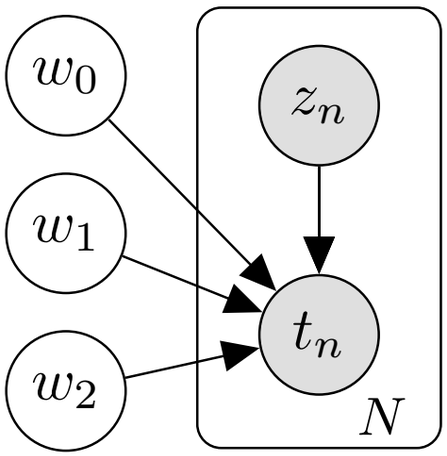
$$\nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_{p(\mathbf{x},\mathbf{y})}[-\nabla_\phi \log q_\phi(\mathbf{x}|\mathbf{y})]$$

No reparameterization required; train on samples from the model. Learns to invert *model.*

Paige, Wood. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML 2016.
Le, Baydin, Wood. Inference Compilation and Universal Probabilistic Programming. AISTATS 2017.

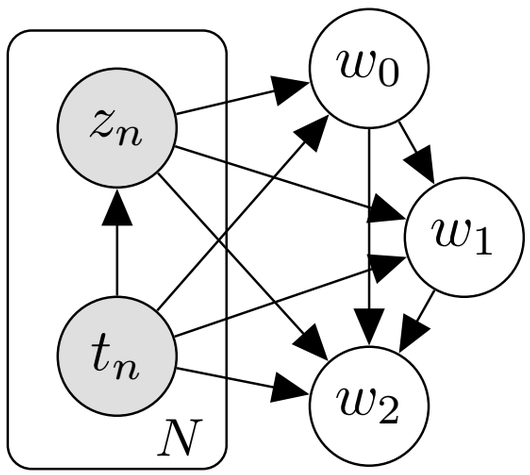
FOPPL Inference Compilation

FOPPL program / graphical model



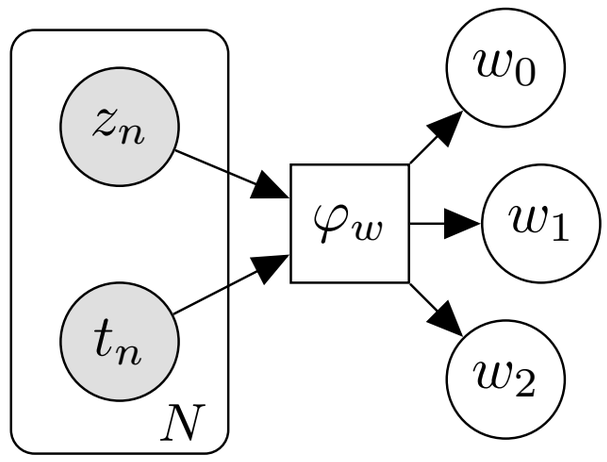
hand

“Inverted” graphical model / course-grain neural-net proposal architecture “stub”



hand

Fine-grain neural-net proposal architecture



$$w_d \sim \text{Laplace}(0, 10^{1-d}) \quad \text{for } d = 0, 1, 2;$$
$$t_n \sim \text{t}_\nu(w_0 + w_1 z_n + w_2 z_n^2, \epsilon^2) \quad \text{for } n = 1, \ldots, N$$

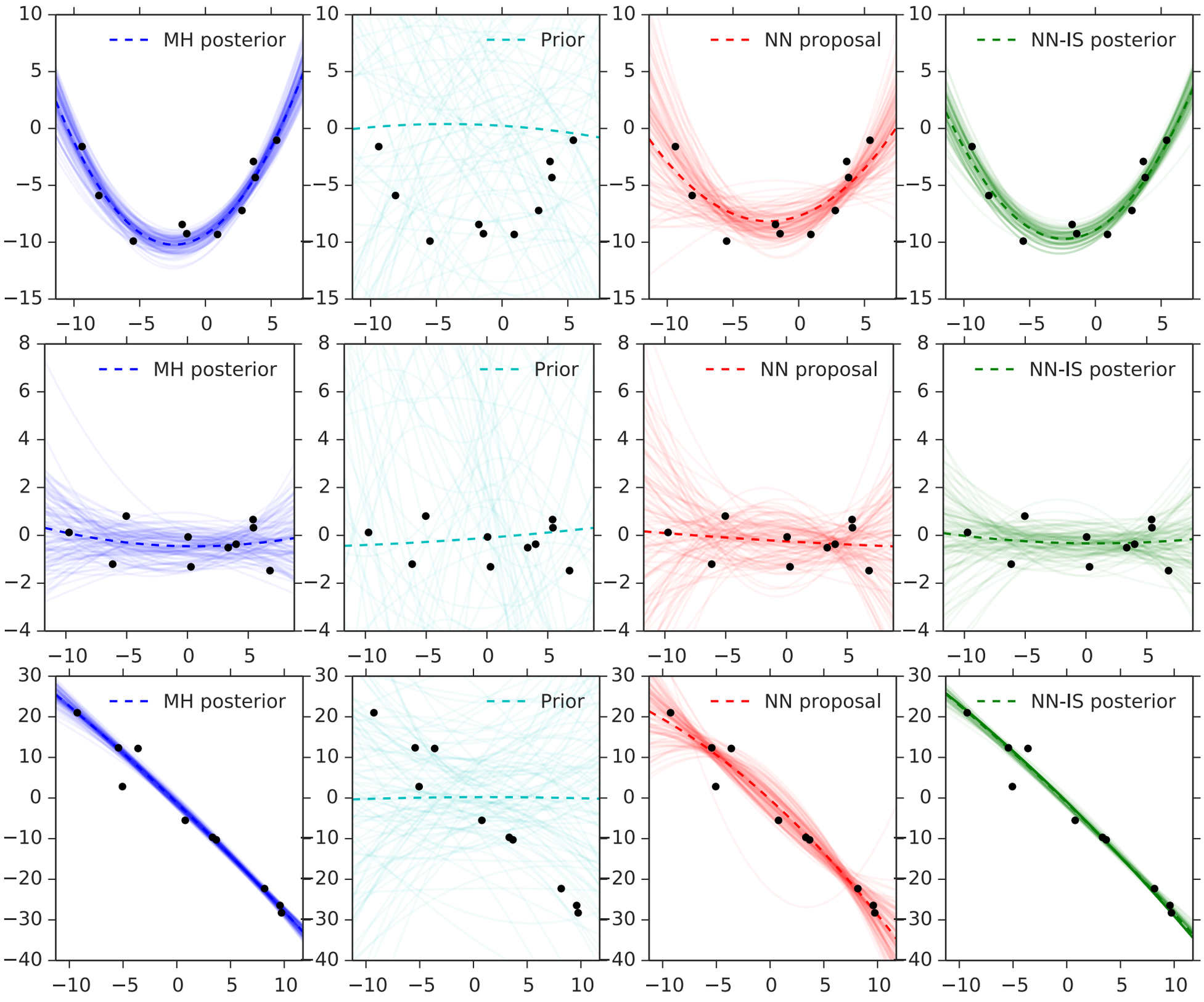
$\nu = 4, \epsilon = 1$, and $z_n \in (-10, 10)$

$$q(w_{0:2} | z_{1:N}, t_{1:N})$$

- Two layer MLP
- 200 units
- 3-component MOG for each output

Paige, Wood. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML 2016.

FOPPL Inference Compilation Example



Paige, Wood. Inference Networks for Sequential Monte Carlo in Graphical Models. ICML 2016.

HOPPL Amortized Inference

- What do we do when there is no probabilistic program / graphical model correspondence?
  - Can't "invert" graphical model
  - Structure of inference network must match any generative model trace "addresses" at runtime
- Options for guide program / inference network:
  - Manually programmed
    - Co-mingle source code with generative model code
    - Require user to ensure matches at addresses
  - Automatic
    - Use a generic but powerful regressor that dynamically generates required addressed values at runtime

“Guide Program” (usually in combo with variational inference)

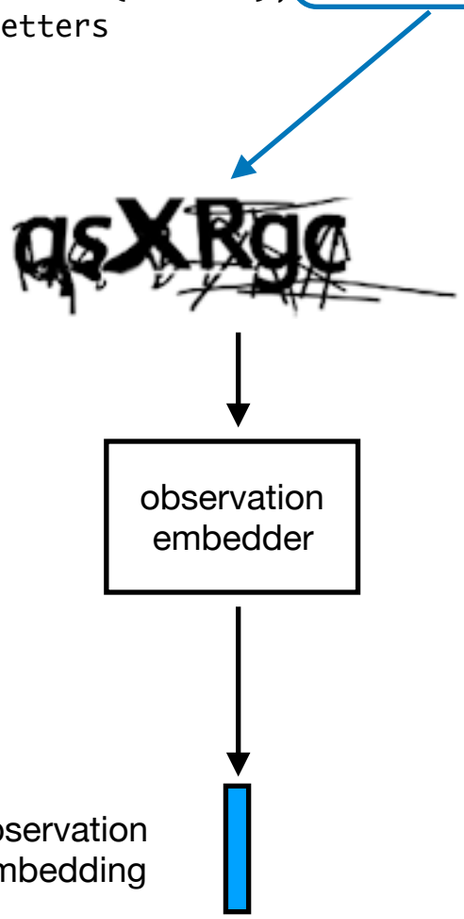
```
// Observed value
var y = 0.5;

// Neural net setup
var nIn = 1;
var nHidden = 3;
var nOut = 2;

var model = function() {
   // Neural net params
   var W1 = param({dims: [nHidden, nIn], name: 'W1'});
   var b1 = param({dims: [nHidden, 1], name: 'b1'});
   var W2 = param({dims: [nOut, nHidden], name: 'W2'});
   var b2 = param({dims: [nOut, 1], name: 'b2'});

   // Use neural net to compute guide params
   var nnInput = Vector([y]);
   var nnOutput = linear(sigmoid(linear(nnInput, W1, b1)), W2, b2);

   var x = sample(Gaussian({mu: 0, sigma: 1}), {
     guide: Gaussian({mu: T.get(nnOutput, 0), sigma: softplus(T.get(nnOutput, 1))})
   });
   observe(Gaussian({mu: x, sigma: 0.5}), y);
   return x;
};
```

- Guide programs
  - same control flow as the original program but a different data flow
  - samples the same random choices as simulator and in the same order; data flowing into those choices comes from different computation.

Ritchie, Daniel, Paul Horsfall, and Noah D. Goodman. "Deep amortized inference for probabilistic programs." *arXiv preprint arXiv:1610.05735* (2016).

HOPPL Inf. Comp. Generic Polymorphic Inference Network

```
letters = []
num_letters = sample(Poisson(6))
for i in range(num_letters):
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```



Le TA, Baydin AG, Wood F. Inference Compilation and Universal Probabilistic Programming. AISTATS 2017.

HOPPL Inf. Comp. Generic Polymorphic Inference Network

```
letters = []
num_letters = sample(Poisson(6))
for i in range(num_letters):
   letters.append(sample(Uniform("a",…,"z","A",…,"Z")))

observe(render(letters), observed_captcha)
return letters
```

observation embedder

observation embedding

sample from the proposal: 6

proposal: … 4 5 6 7 8 9 …

Proposal layer

LSTM time step 1

previous sample embedding: 0

address: a_1

instance: 1

type: UniformDiscrete

HOPPL Inf. Comp. Generic Polymorphic Inference Network

```
letters = []
num_letters = sample(Poisson(6))
for i in range(num_letters):                          // i = 0
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```

observation embedder

observation embedding

sample from the proposal:

proposal:

… 4 5 6 7 8 9 …

6

“q”

"q" "a" "o"

Proposal layer

Proposal layer

LSTM time step 1

LSTM time step 2

previous sample embedding: 0 6

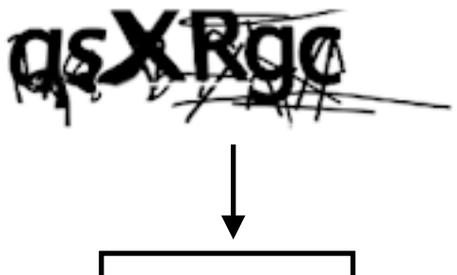
address: a_1 a_2

instance: 1 1

type: UniformDiscrete Uniform

HOPPL Inf. Comp. Generic Polymorphic Inference Network

```
letters = []
num_letters = sample(Poisson(6))
for i in range(num_letters):                          // i = 1
   letters.append(sample(Uniform("a",…,"z","A",…,"Z")))

observe(render(letters), observed_captcha)
return letters
```



observation embedder

observation embedding

sample from the proposal: "q" "s"

proposal: "q" "a" "o" "s" "S"

Proposal layer

Proposal layer

...

LSTM time step 2

LSTM time step 3

previous sample embedding: 6 "q"

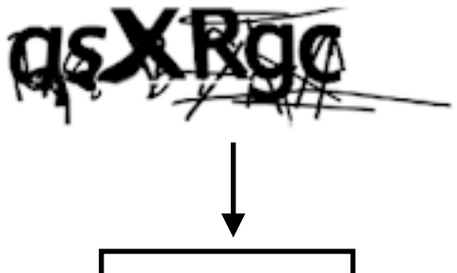
address: a_2 a_2

instance: 1 2

type: Uniform Uniform

HOPPL Inf. Comp. Generic Polymorphic Inference Network

```
letters = []
num_letters = sample(Poisson(6))
for i in range(num_letters):                          // i = 2
   letters.append(sample(Uniform("a",…,"z","A",…,"Z")))

observe(render(letters), observed_captcha)
return letters
```



observation embedder

observation embedding

sample from the proposal:

proposal:

"s" "S"

"X"

"s"

"X"

Proposal layer

Proposal layer

...

LSTM time step 3

LSTM time step 4

previous sample embedding:

"q"

"s"

address:

a_2

a_2

instance:

2

3

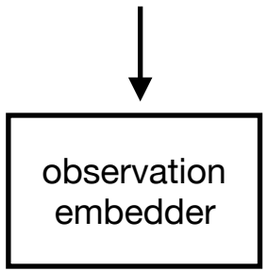
type:

Uniform

Uniform

HOPPL Inf. Comp. Generic Polymorphic Inference Network

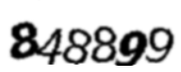
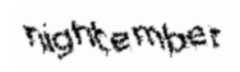
```
letters = []
num_letters = sample(Poisson(6))
for i in range(num_letters):                          // i = …
   letters.append(sample(Uniform(“a”,…,”z”,“A”,…,”Z”)))

observe(render(letters), observed_captcha)
return letters
```



…

etc.

HOPPL Inf. Comp. Example : Captcha Breaking

Type		Baidu (2011) 	Baidu (2013) 	eBay 	Yahoo 	reCaptcha 	Wikipedia 	Facebook 
Our method	RR	99.8%	99.9%	99.2%	98.4%	96.4%	93.6%	91.0%
	BT	72 ms	67 ms	122 ms	106 ms	78 ms	90 ms	90 ms
Bursztein et al. [15]	RR	38.68%	55.22%	51.39%	5.33%	22.67%	28.29%	
	BT	3.94 s	1.9 s	2.31 s	7.95 s	4.59 s		
Starostenko et al. [16]	RR				91.5%	54.6%		
	BT					< 0.5 s		
Gao et al. [17]	RR	34%			55%	34%		
Gao et al. [18]	RR		51%		36%			
	BT		7.58 s		14.72 s			
Goodfellow et al. [6]	RR					99.8%		
Stark et al. [8]	RR					90%		

Facebook Captcha

Observed images		 (W4kgvQ)	 (uV7FeWB)	 (MqhnpT)
Inference				
Training traces	10^7	W4kgvQ	uV7EeWB	MqhnpT
	10^6	WA4rjvQ	uV7FeWB	MypppT
	10^5	Woxewd9	mTTEMMm	RIrpES
	10^4	BKvu2Q	C9QDsoN	rS5FP2B

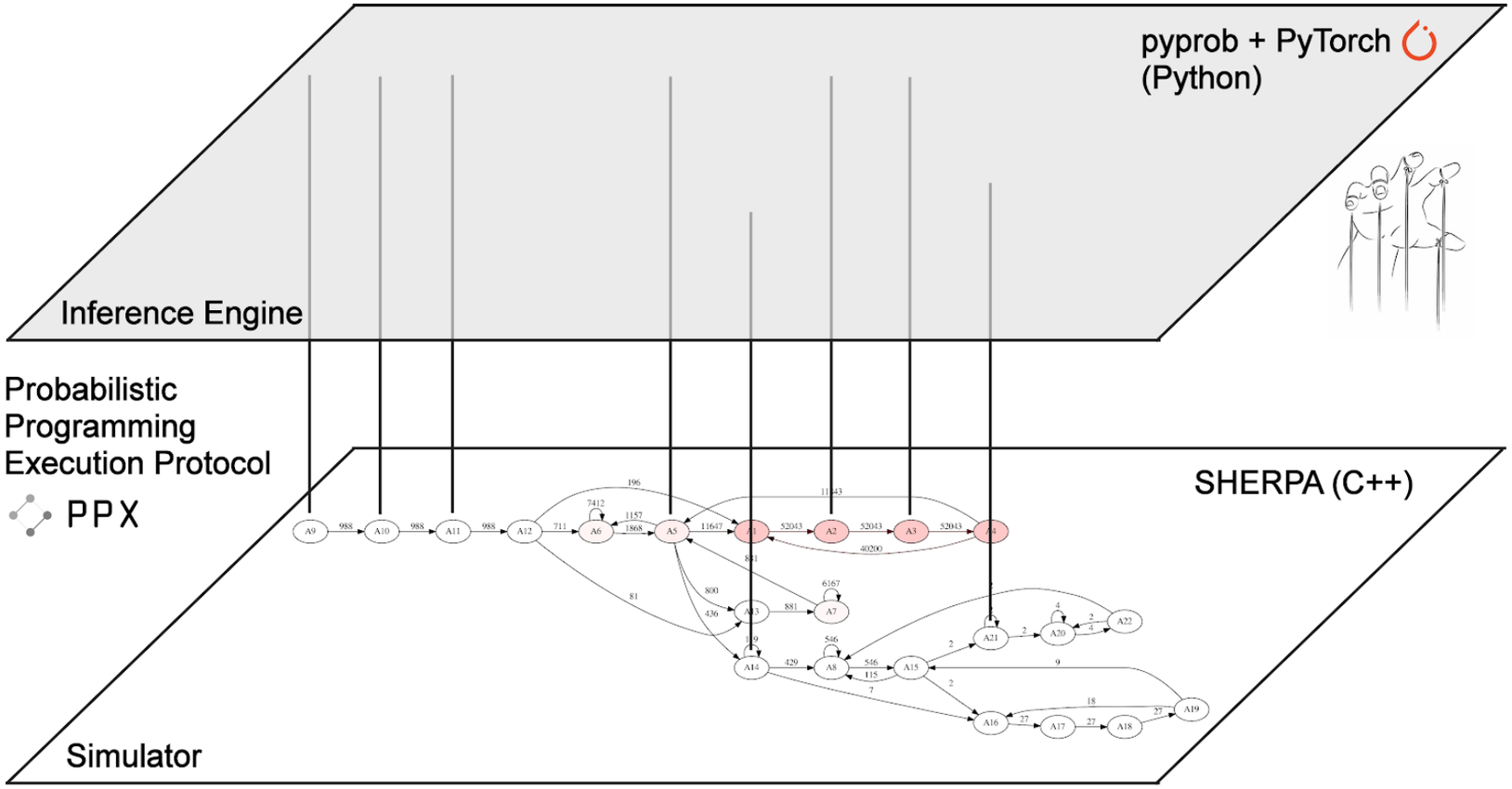
vicarious

$40M raise

Le TA, Baydin AG, Wood F. Inference Compilation and Universal Probabilistic Programming. AISTATS 2017.

Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model

Atılım Güneş Baydin,[1] **Lukas Heinrich,**[2] **Wahid Bhimji,**[3]
Bradley Gram-Hansen,[1] **Gilles Louppe,**[4] **Lei Shao,**[5]
Prabhat,[3] **Kyle Cranmer,**[2] **Frank Wood**[6]
[1]University of Oxford; [2]New York University; [3]Lawrence Berkeley National Lab
[4]University of Liège; [5]Intel Corporation; [6]University of British Columbia
{gunes,bradley}@robots.ox.ac.uk; {lukas.heinrich,g.louppe,cranmer}@cern.ch
{wbhimji,prabhat}@lbl.gov; lei.shao@intel.com; fwood@cs.ubc.ca



probprog / pyprob

Unwatch 30 | Star 176 | Fork 17

Code | Issues 4 | Pull requests 1 | Projects 0 | Wiki | Insights | Settings

A PyTorch-based library for probabilistic programming and inference compilation

Edit

Manage topics

837 commits | 6 branches | 1 release | 5 contributors | BSD-2-Clause

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

gbaydin Update — Latest commit 2889e40 2 days ago

pyprob	Update	2 days ago
tests	New Empirical distribution with disk support	7 days ago
.gitignore	Merge branch 'master' into teng-dev	7 months ago
.travis.yml	Analytics bug fix	14 days ago
Dockerfile	Add gdbm to Dockerfile	6 days ago
LICENSE	Create LICENSE	3 months ago
README.md	Update README.md	3 months ago
setup.py	Start adding PPX	16 days ago

README.md

pyprob

build passing

`pyprob` is a PyTorch-based library for probabilistic programming and inference compilation. The main focus of this library is on coupling existing simulation codebases with probabilistic inference with minimal intervention.

High Peaks -- Our Battle to Control a SHERPA

- Made using **PyProb** (find on GitHub)
  - 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

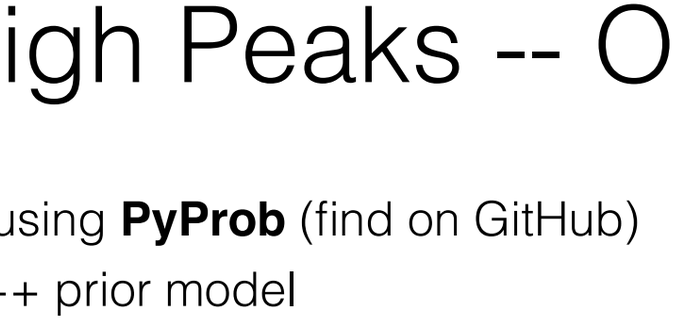
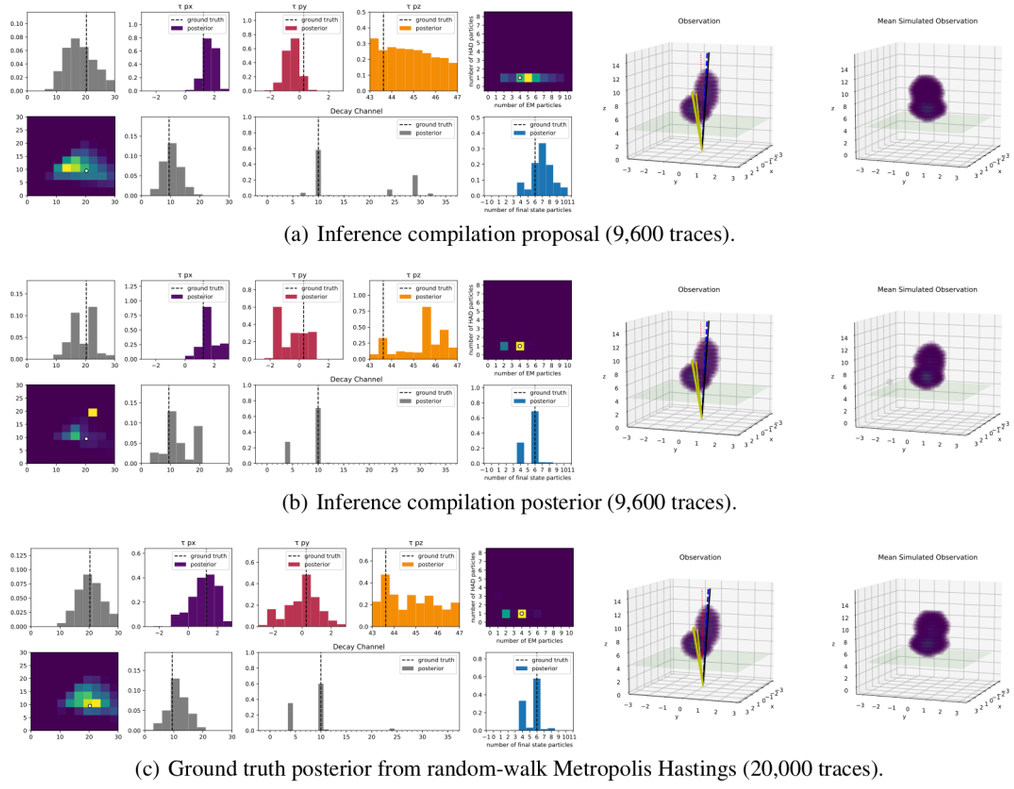


Figure 1: *Top:* branching ratios of the tau lepton, effectively the prior distribution of the decay channels in the SHERPA simulation. Note that the scale is logarithmic. *Bottom:* Feynman diagrams for tau decays illustrating these can produce multiple detected particles.

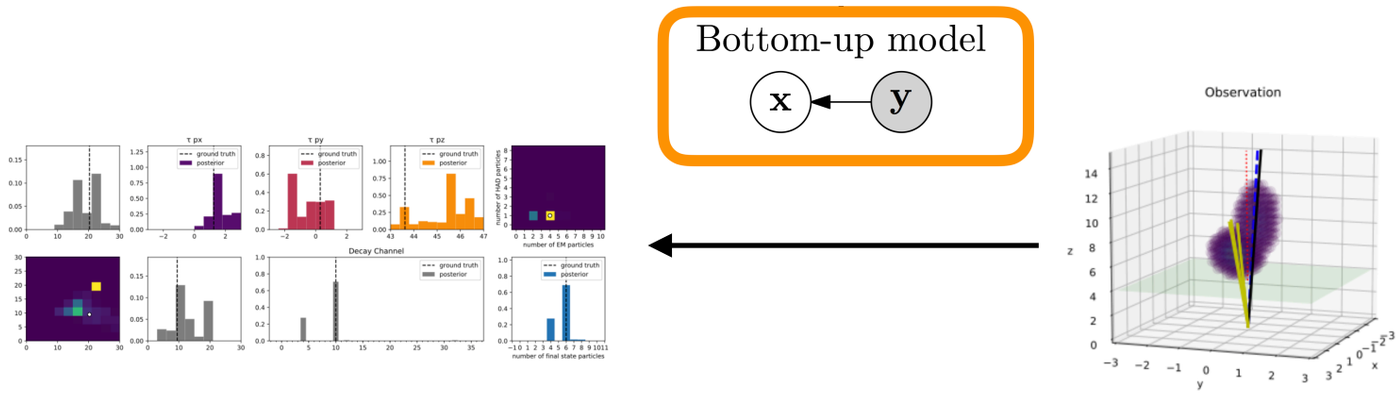


(a) Inference compilation proposal (9,600 traces).

(b) Inference compilation posterior (9,600 traces).

(c) Ground truth posterior from random-walk Metropolis Hastings (20,000 traces).

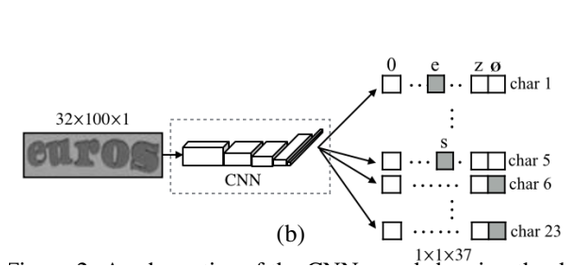
Figure 2: The corner plot on the left, shows the particle energies of the two most energetic final state particles and their joint probability. To the right, the distribution of the originating momentum components of the τ lepton and its decay mode is shown. In the middle we show the event composition as characterized by the number of mainly 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.



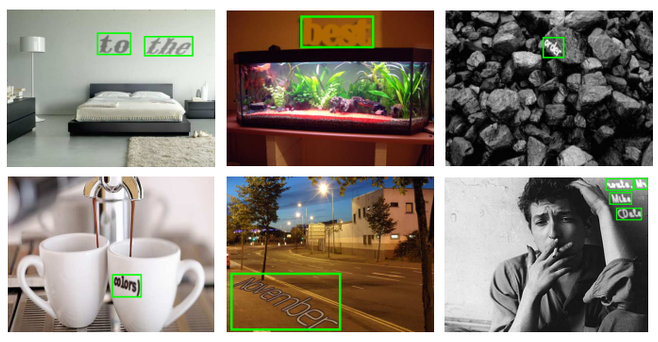
Using Synthetic Data to Train Neural Networks is Model-Based Reasoning

Tuan Anh Le*, Atılım Güneş Baydin*, Robert Zinkov† and Frank Wood*
*Department of Engineering Science
University of Oxford, Parks Road, OX1 3PJ Oxford, UK
Email: {tuananh, gunes, fwood}@robots.ox.ac.uk
†School of Informatics and Computing
Indiana University, 919 E 10th Street, Bloomington, IN 47408, USA
Email: zinkov@iu.edu

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.

Pros vs. Cons of Inference Compilation

- Pros
  - Infinite training data
  - No reparameterization or control variates required
  - Can perform inference *in* pre-existing simulators
- Cons
  - Choice of “embedding” architecture is largely ad hoc but very important
  - Must run generative model program during inference; inference computational complexity is dictated by generative model computational complexity
    - Surrogate models

Determining neural-net structure is hard

Inference Network Structure From Generative Model

- Can we determine inference network structure from generative model structure?
  - i.e. "stub out" inference network given the generative model

- *Yes*, in FOPPL context

Automatic Derivation of Encoder Architecture

Faithful Inversion of Generative Models for Effective Amortized Inference

Stefan Webb[*] University of Oxford
Adam Goliński University of Oxford
Robert Zinkov University of Oxford
N. Siddharth University of Oxford
Tom Rainforth University of Oxford
Yee Whye Teh University of Oxford
Frank Wood University of British Columbia

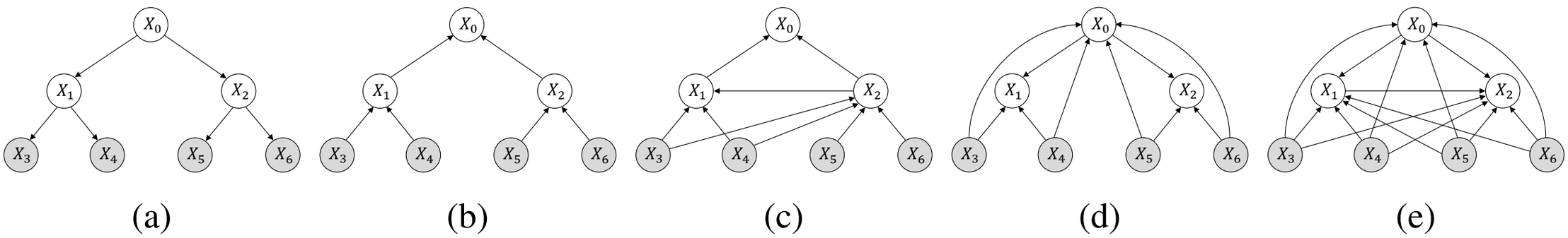
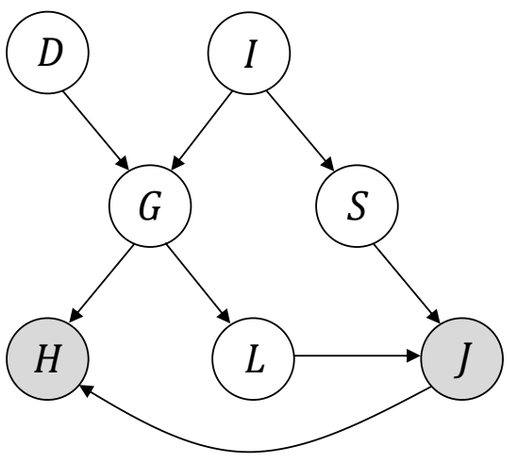


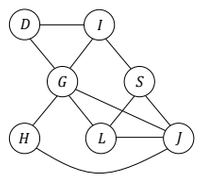
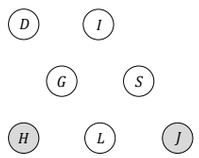
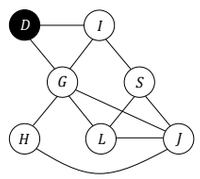
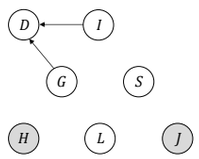
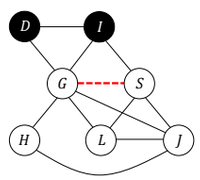
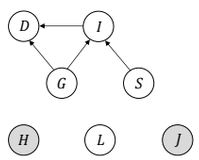
Figure 4: (a) BN structure for a binary tree with $d = 3$; (b) Stuhlmüller's heuristic inverse; (c) Natural minimally faithful inverse produced by NaMI in topological mode; (d) Most compact inverse when $d > 3$, given by running NaMI in reverse topological mode; (e) Fully connected inverse.

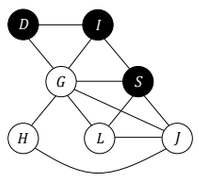
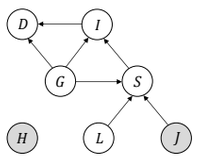
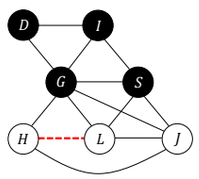
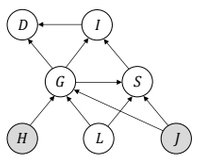
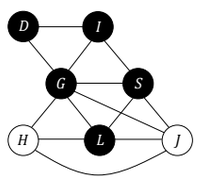
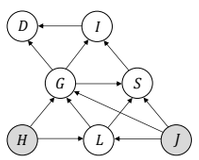
NaMI Algorithm

- Constructs an inverse graphical model structure that is provably both
  - *Faithful* : contains sufficient edges to avoid encoding conditional independencies absent from the model.
  - *Minimal* : does not contain any unnecessary edges; i.e., removing any edge would result in an unfaithful structure.

- Works by simulating variable elimination



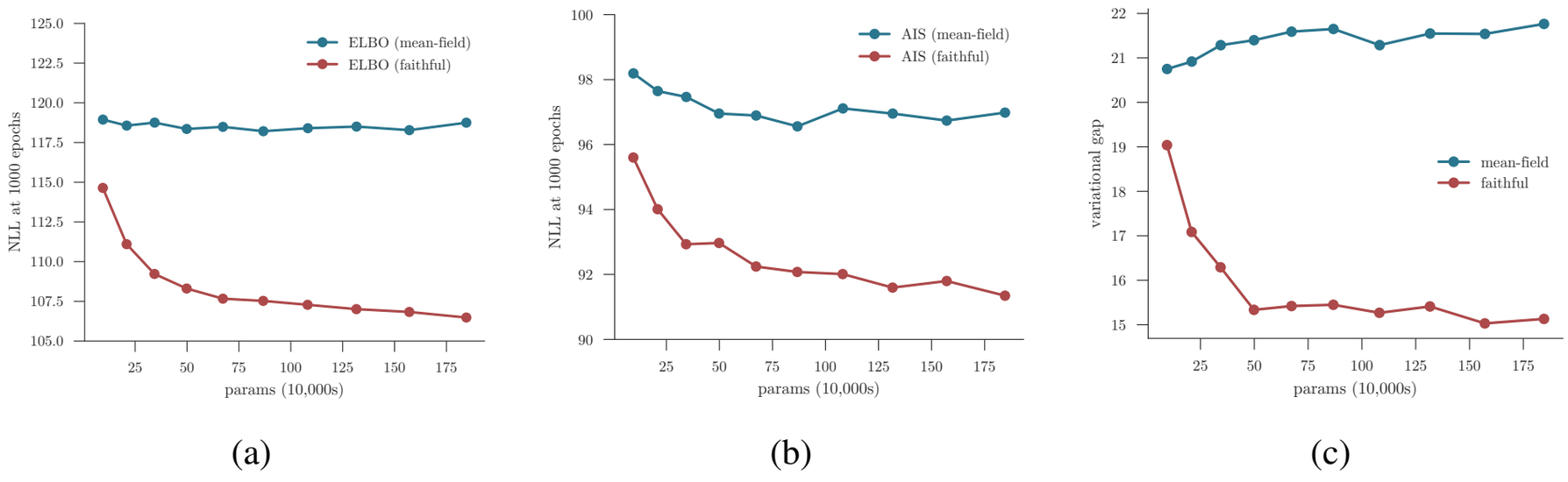
Step	S	v	$\mathcal{J}$	$\mathcal{H}$
0	—	—		
1	D, I	D		
2	I	I		

Step	S	v	$\mathcal{J}$	$\mathcal{H}$
3	G, S	S		
4	G	G		
5	L	L		

Webb et al. Faithful Inversion of Generative Models for Effective Amortized Inference. NIPS 2018

Lessons

- Course-grain structure matters for both guide programs and inference networks
  - The wrong course-grain structure hurts no matter what the fine-grain capacity



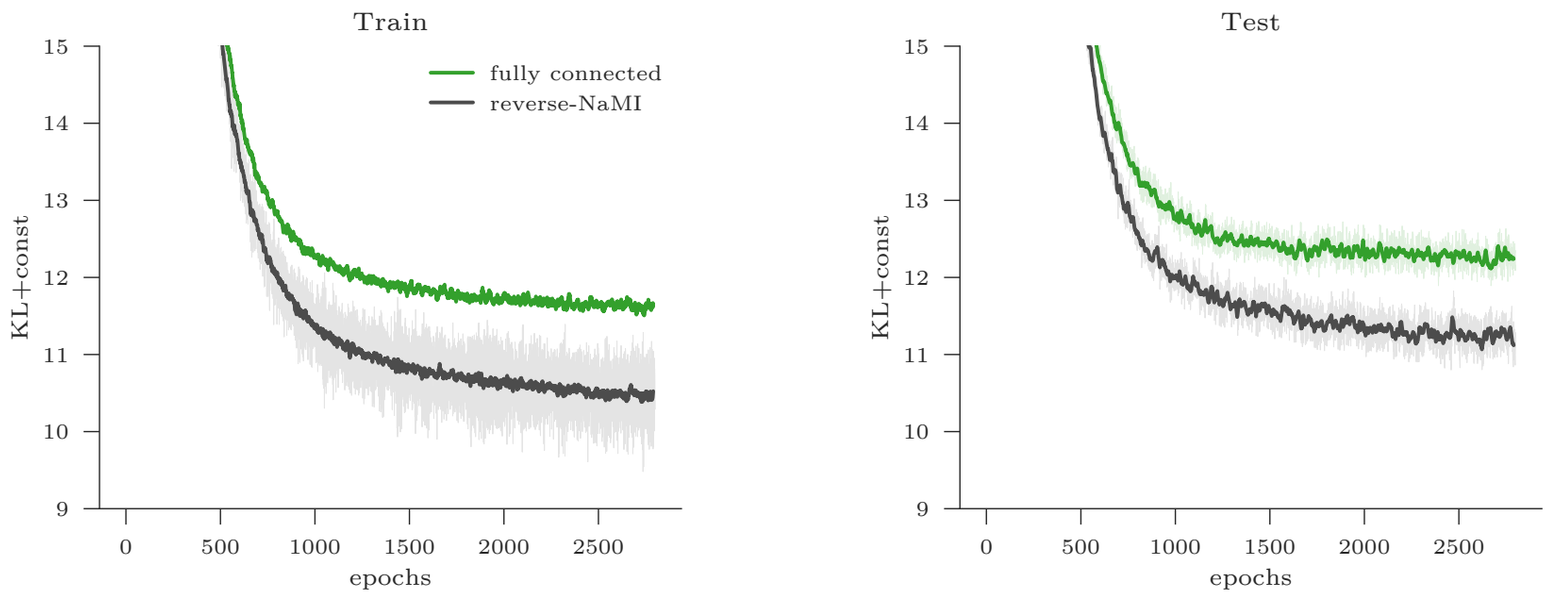
(a) (b) (c)

Figure 3: Results for the relaxed Bernoulli VAE with 30 latent units, compared after 1000 epochs of learning the: (a) negative ELBO, and (b) negative AIS estimates, varying inference network factorizations and capacities (total number of parameters); (c) An estimate of the variational gap, that is, the difference between marginal log-likelihood and the ELBO.

Webb et al. Faithful Inversion of Generative Models for Effective Amortized Inference. NIPS 2018

Lessons

- Course-grain structure matters for both guide programs and inference networks
  - The wrong course-grain structure hurts no matter what the fine-grain capacity



(a) (b)

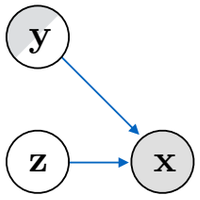
Figure 6: Results for Bayesian GMM for $K = 3$ clusters and $N = 200$ data points, comparing inference networks in the compiled inference setting. The reverse KL divergence objective estimated on the training and test sets are shown in (a) and (b) respectively. The shaded regions indicate 1 standard error.

Webb et al. Faithful Inversion of Generative Models for Effective Amortized Inference. NIPS 2018

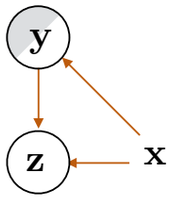
Writing Models Is Hard

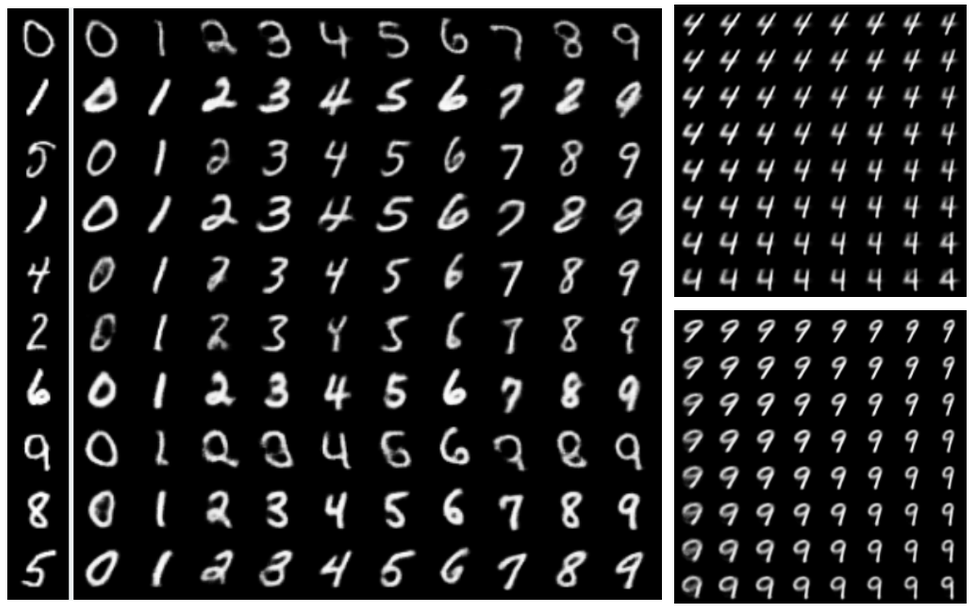
ProbTorch/Pyro : Jointly Learn Model and Guide Program

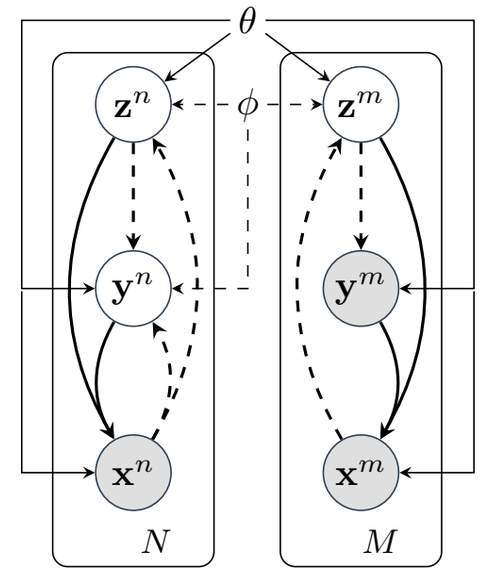
Generative Model

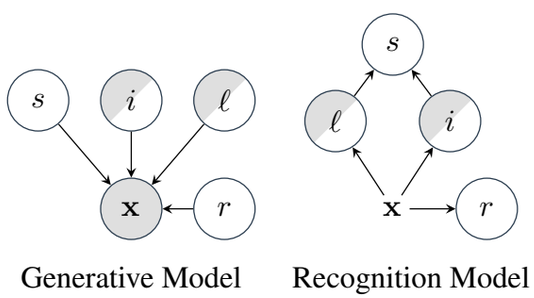


Recognition Model









Generative Model Recognition Model

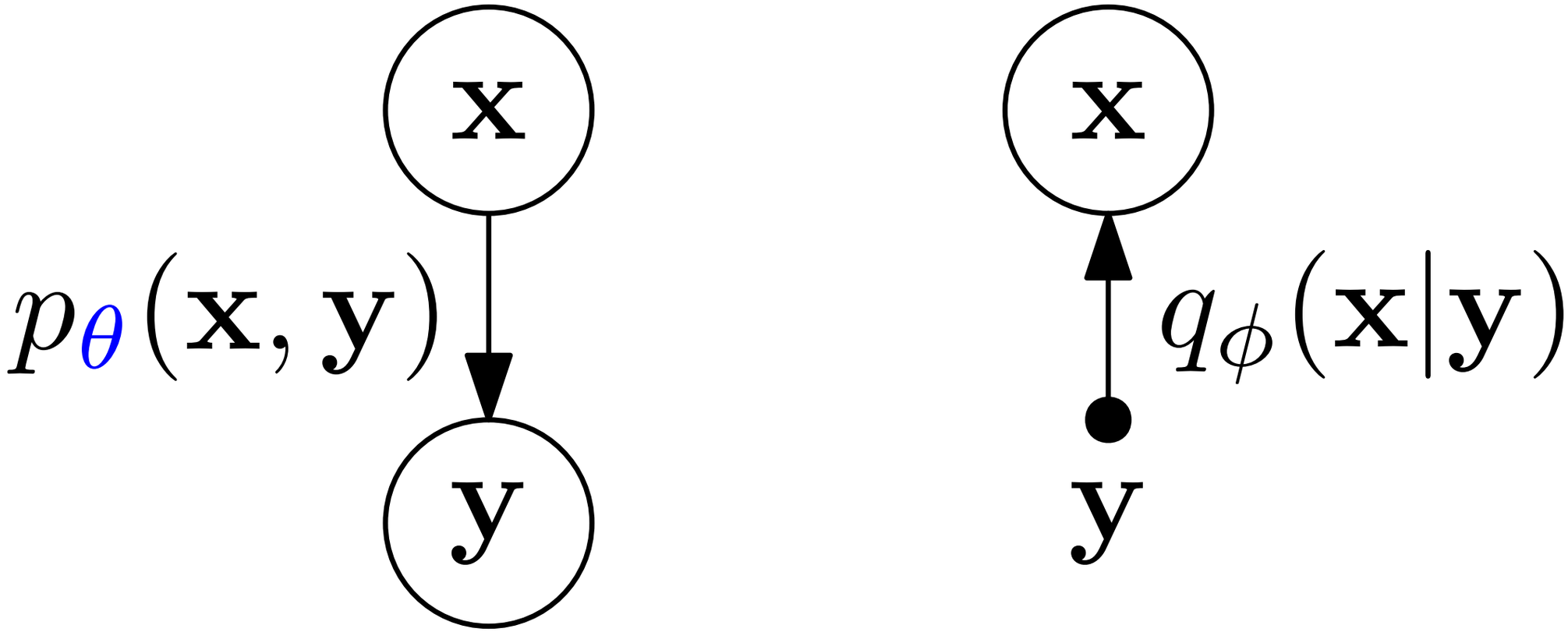
PROB TORCH

PYRO
Deep Universal Probabilistic Programming

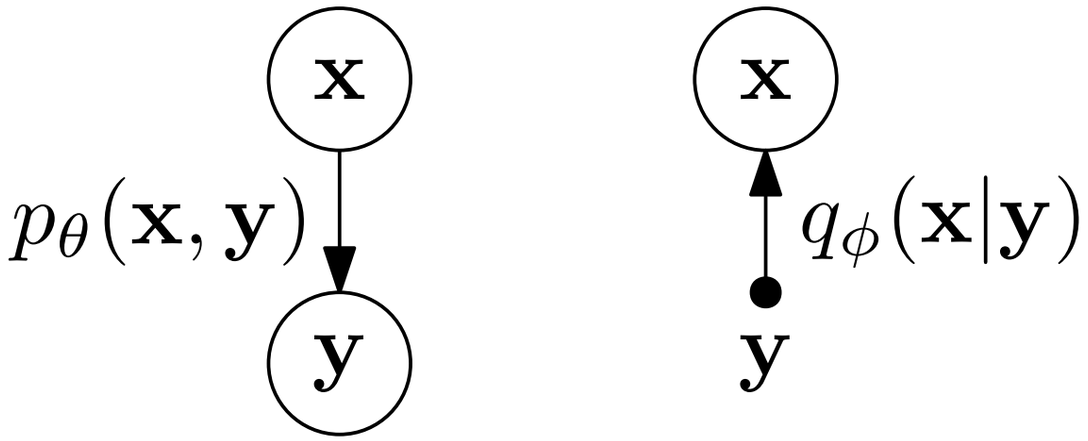
Edward

Siddarth, Paige, van de Meent, Desmaison, Goodman, Kohli, Wood, Torr
Learning Disentangled Representations with Semi-Supervised Deep Generative Models NIPS 2017

Simultaneous Model and Guide Program Parameter Learning



Structured Variational Autoencoders



$$\mathrm{ELBO}_{\mathrm{VAE}}(\theta, \phi, \mathbf{y}) = \log p_\theta(\mathbf{y}) - \mathrm{KL}(q_\phi(\mathbf{x}|\mathbf{y})||p_\theta(\mathbf{x}|\mathbf{y}))$$

$$= \mathbb{E}_{q_\phi(\mathbf{x}|\mathbf{y})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{y})}{q_\phi(\mathbf{x}|\mathbf{y})}\right]$$

Kingma, Welling. Auto-encoding variational Bayes. ICLR 2014.
Rezende, Mohamed, Wierstra. ICML 2014.

Research Ideas

- Can we use more computation to make this bound tighter and do an even better job learning the model and inference network?
- Can we accommodate lazy evaluation and branching on non-determinism in a way that doesn't require inefficient reinforce-style control-variate schemes? (think discrete random variables)

Importance-Weighted Autoencoders

$$\mathrm{ELBO}_{\mathrm{VAE}}(\theta, \phi, \mathbf{y}) = \mathbb{E}_{q_\phi(\mathbf{x}|\mathbf{y})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{y})}{q_\phi(\mathbf{x}|\mathbf{y})}\right]$$

importance sampling

$$\mathrm{ELBO}^K_{\mathrm{IS}}(\theta, \phi, \mathbf{y}) = \mathbb{E}_{q_\phi(\mathbf{x}_1|\mathbf{y})\cdots q_\phi(\mathbf{x}_K|\mathbf{y})}\left[\log\left(\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(\mathbf{x}_k, \mathbf{y})}{q_\phi(\mathbf{x}_k|\mathbf{y})}\right)\right]$$

Burda, Grosse, Salakhutdinov. Importance weighted autoencoders. ICLR 2016.

Auto-encoding Sequential Monte Carlo

$$\mathrm{ELBO}_{\mathrm{VAE}}(\theta, \phi, \mathbf{y}) = \mathbb{E}_{q_\phi(\mathbf{x}|\mathbf{y})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{y})}{q_\phi(\mathbf{x}|\mathbf{y})}\right]$$

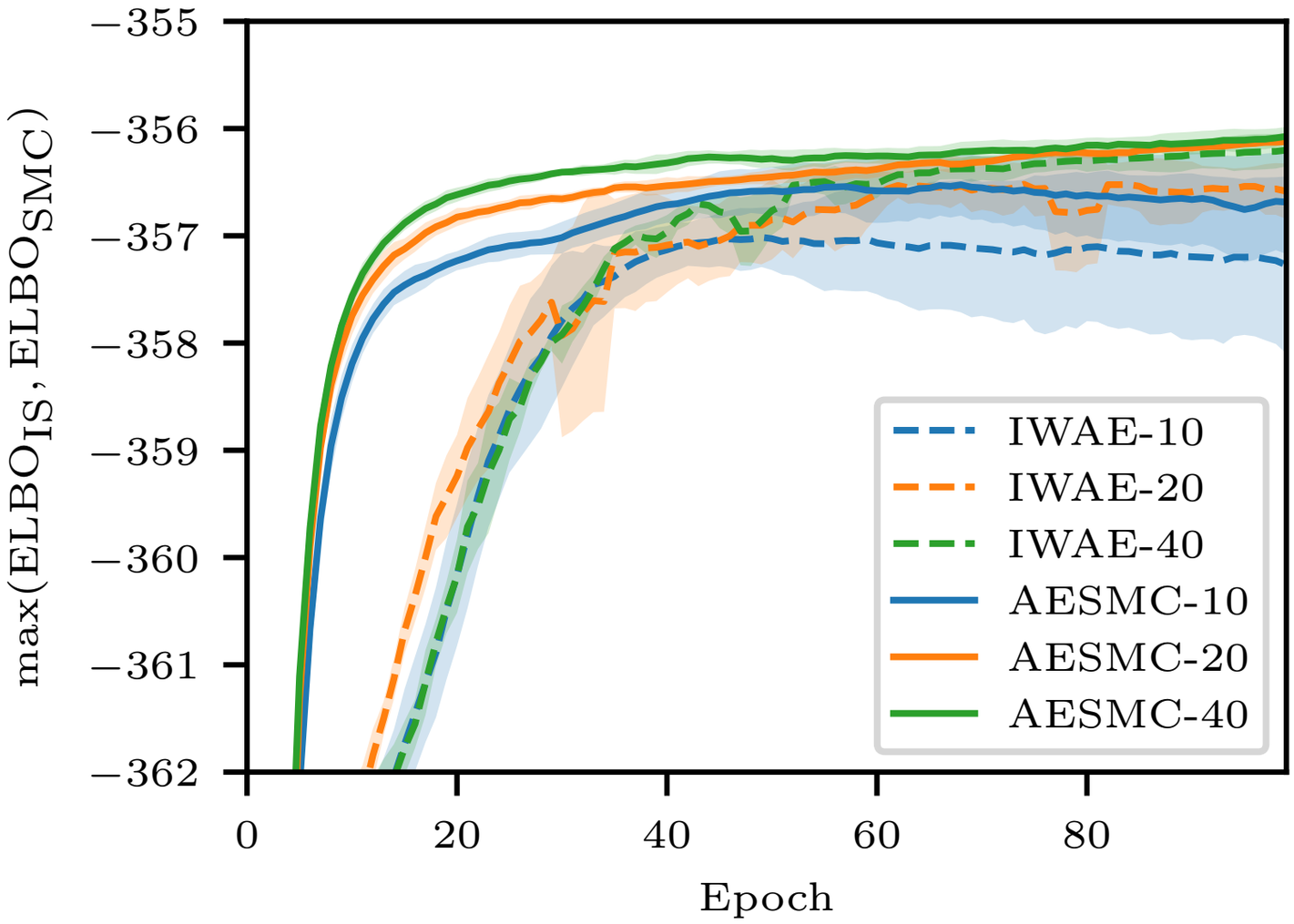
any sample-based marginal likelihood estimator

importance sampling

$$\mathrm{ELBO}^K_{\mathrm{IS}}(\theta, \phi, \mathbf{y}) = \mathbb{E}_{q_\phi(\mathbf{x}_1|\mathbf{y}) \cdots q_\phi(\mathbf{x}_K|\mathbf{y})}\left[\log \left(\hat{Z}_{\theta,\phi}(\mathbf{y}) \frac{1}{K} \sum_{k=1}^{K} \right) \frac{p_\theta(\mathbf{x}_k, \mathbf{y})}{q_\phi(\mathbf{x}_k|\mathbf{y})}\right]$$

Maddison, et al. Filtering variational objectives. NIPS 2017.
Le, Igl, Rainforth, Jin, Wood. Auto-encoding sequential Monte Carlo. ICLR 2018.
Naesseth, et al. Variational sequential Monte Carlo. AISTATS 2018.

Auto-Encoding Sequential Monte Carlo

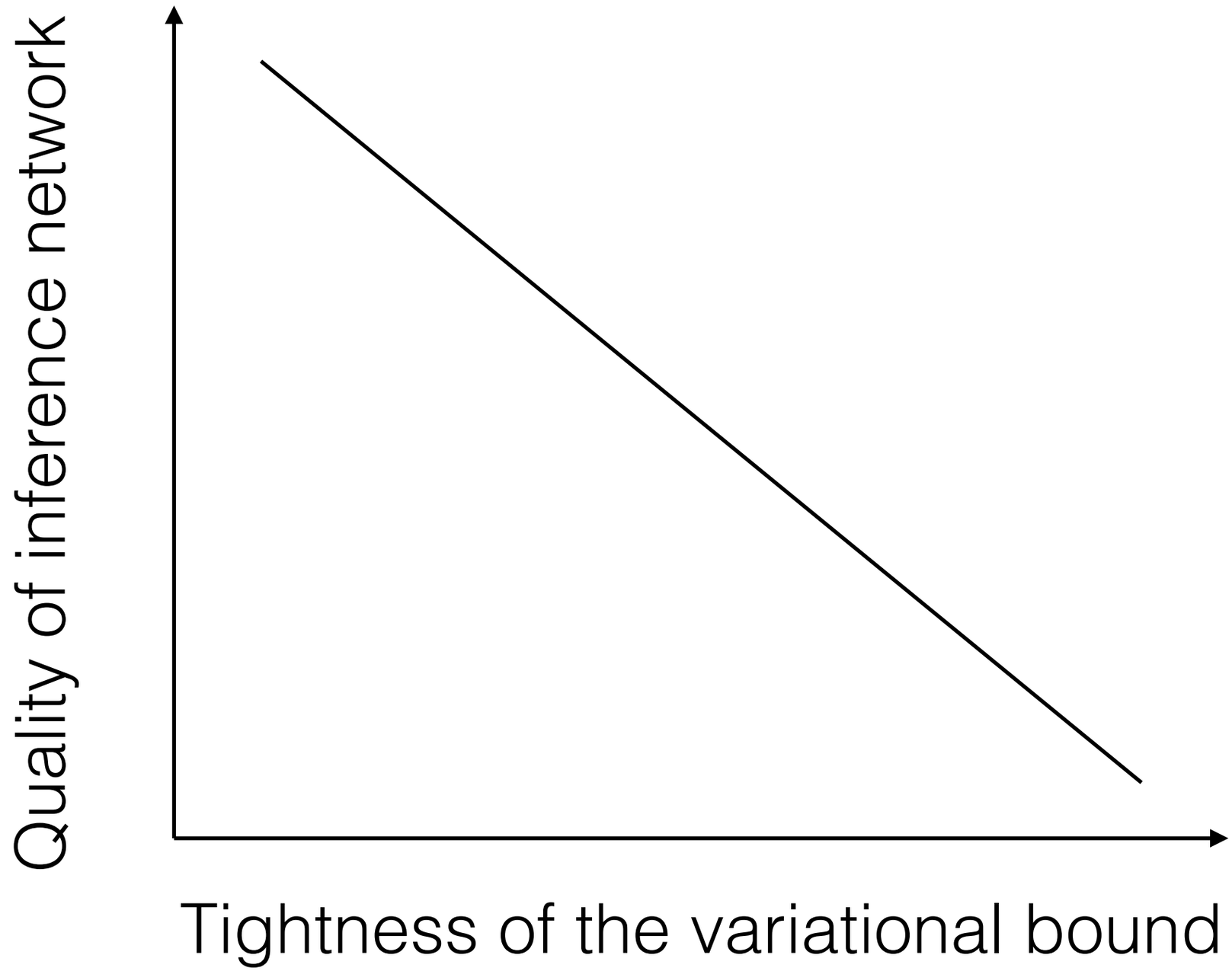


Maddison, et al. Filtering variational objectives. NIPS 2017.

Le, Igl, Rainforth, Jin, Wood. Auto-encoding sequential Monte Carlo. ICLR 2018.

Naesseth, et al. Variational sequential Monte Carlo. AISTATS 2018.

Gotcha: Tighter variational bounds are not necessarily better



Rainforth, Kosiorek, Le, Maddison, Igl, Wood, Teh. Tighter variational bounds. ICML 2018.

Revisiting Reweighted Wake-Sleep

Wake-phase loss for model parameters

$$\mathbb{E}_{p_{\text{true}}}(\mathbf{y}) \left[-\text{ELBO}_{\text{IS}}^{K}(\theta, \phi, \mathbf{y}) \right]$$

Sleep-phase loss for inference network parameters

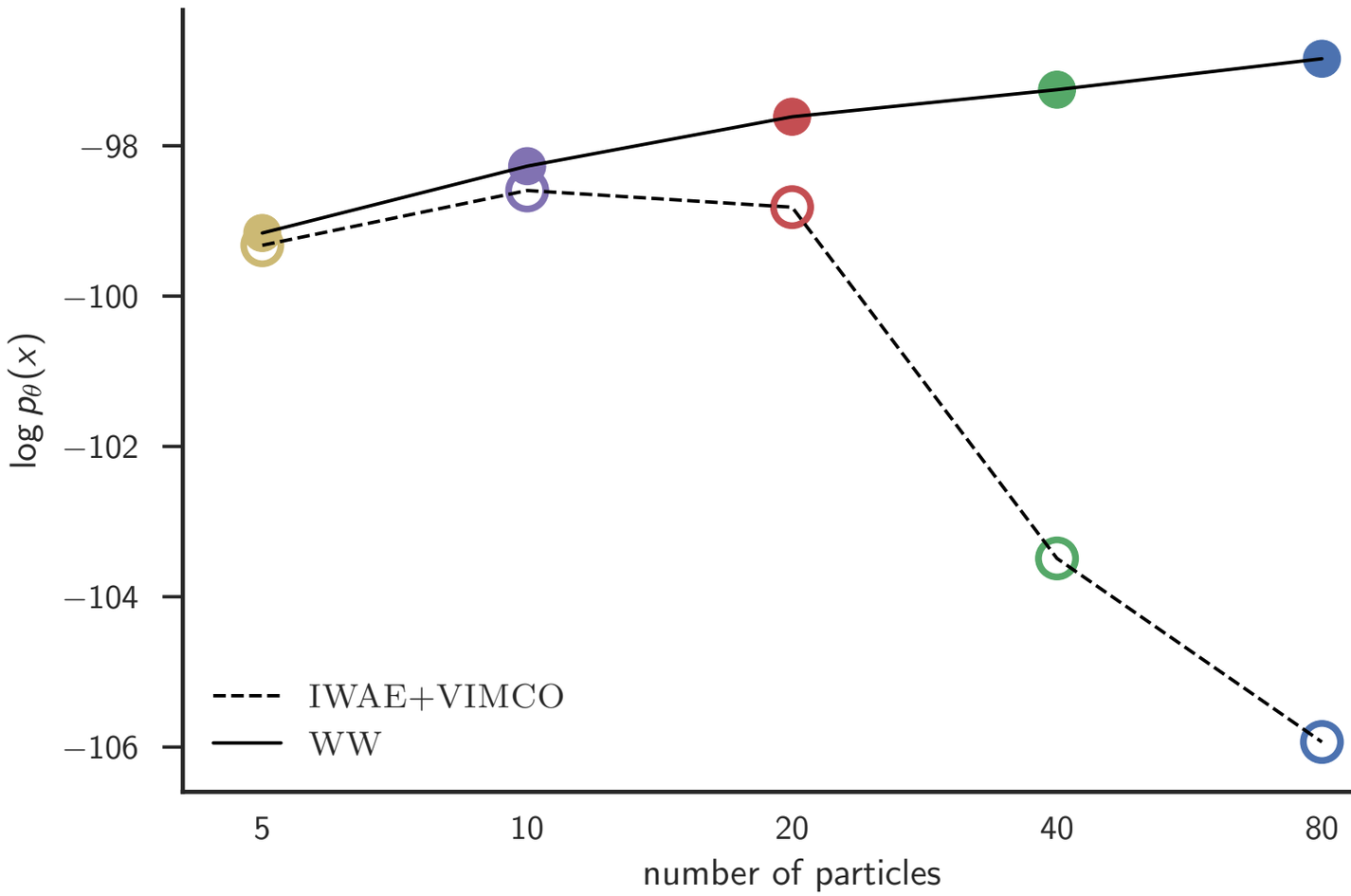
$$\mathbb{E}_{p_{\theta}(\mathbf{y})} \left[\text{KL}(p_{\theta}(\mathbf{x}|\mathbf{y}) || q_{\phi}(\mathbf{x}|\mathbf{y})) \right]$$

Wake-phase loss for inference network parameters

$$\mathbb{E}_{p_{\text{true}}(\mathbf{y})} \left[\text{KL}(p_{\theta}(\mathbf{x}|\mathbf{y}) || q_{\phi}(\mathbf{x}|\mathbf{y})) \right]$$

Bornschein, Bengio. Reweighted wake-sleep. ICLR 2015.
Le*, Kosiorek*, Siddharth, Teh, Wood. Revisiting Reweighted Wake-Sleep. In submission. 2018.

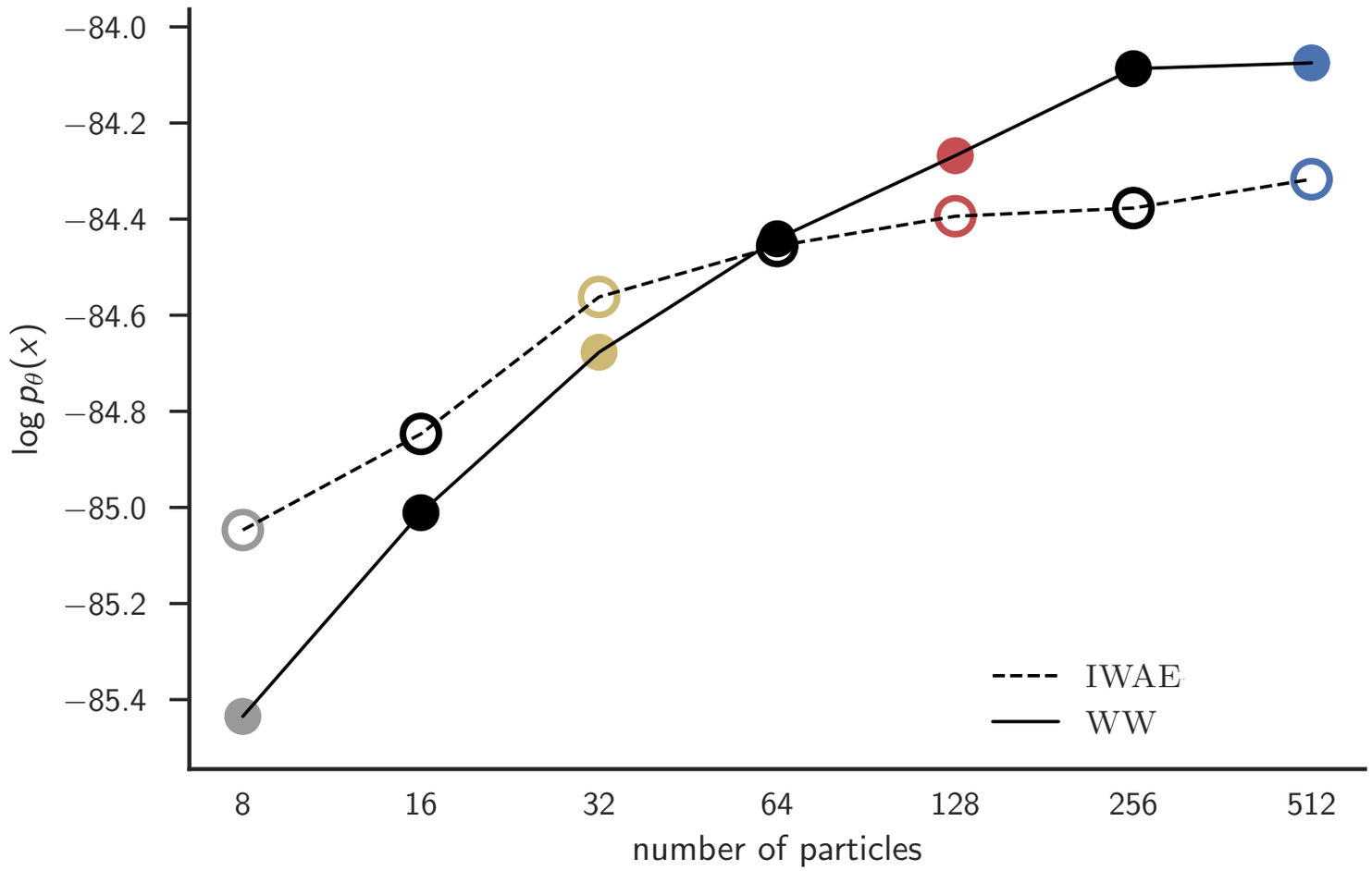
Attend-Infer-Repeat



Eslami et al. Attend, infer, repeat. NIPS 2016.

Le*, Kosiorek*, Siddharth, Teh, Wood. Revisiting Reweighted Wake-Sleep. In submission. 2018.

Continuous Latent Variable Model of MNIST



Le*, Kosiorek*, Siddharth, Teh, Wood. Revisiting Reweighted Wake-Sleep. In submission. 2018.

Thoughts

- Large numbers of models *already exist* as stochastic simulators written in languages other than Python/PyTorch
  - Probabilistic programming inference techniques should work in this setting too
  - Want to do inference *in* said model using existing code
  - Helpful to think about a probabilistic program as an implicit model from the OS perspective as a black box that calls rand, randn, and writes out a result?

- Problems common to all approaches to amortized inference
  - Proposal family?
    - Structure of neural network?
  - The posterior distribution at a particular address is usually not in the family as the prior, how do we programmatically deal with this?

- We all make up ABC-style likelihoods then realize that sharp likelihoods are horrible; should we really be thinking about doing GAN-style inference?

Extra : New Prob Prog Book

arXiv.org > stat > arXiv:1809.10756

Search or Arti

(Help | Advanced s

Statistics > Machine Learning

An Introduction to Probabilistic Programming

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, Frank Wood

(Submitted on 27 Sep 2018)

This document is designed to be a first-year graduate-level introduction to probabilistic programming. It not only provides a thorough background for anyone wishing to use a probabilistic programming system, but also introduces the techniques needed to design and build these systems. It is aimed at people who have an undergraduate-level understanding of either or, ideally, both probabilistic machine learning and programming languages.

We start with a discussion of model-based reasoning and explain why conditioning as a foundational computation is central to the fields of probabilistic machine learning and artificial intelligence. We then introduce a simple first-order probabilistic programming language (PPL) whose programs define static-computation-graph, finite-variable-cardinality models. In the context of this restricted PPL we introduce fundamental inference algorithms and describe how they can be implemented in the context of models denoted by probabilistic programs.

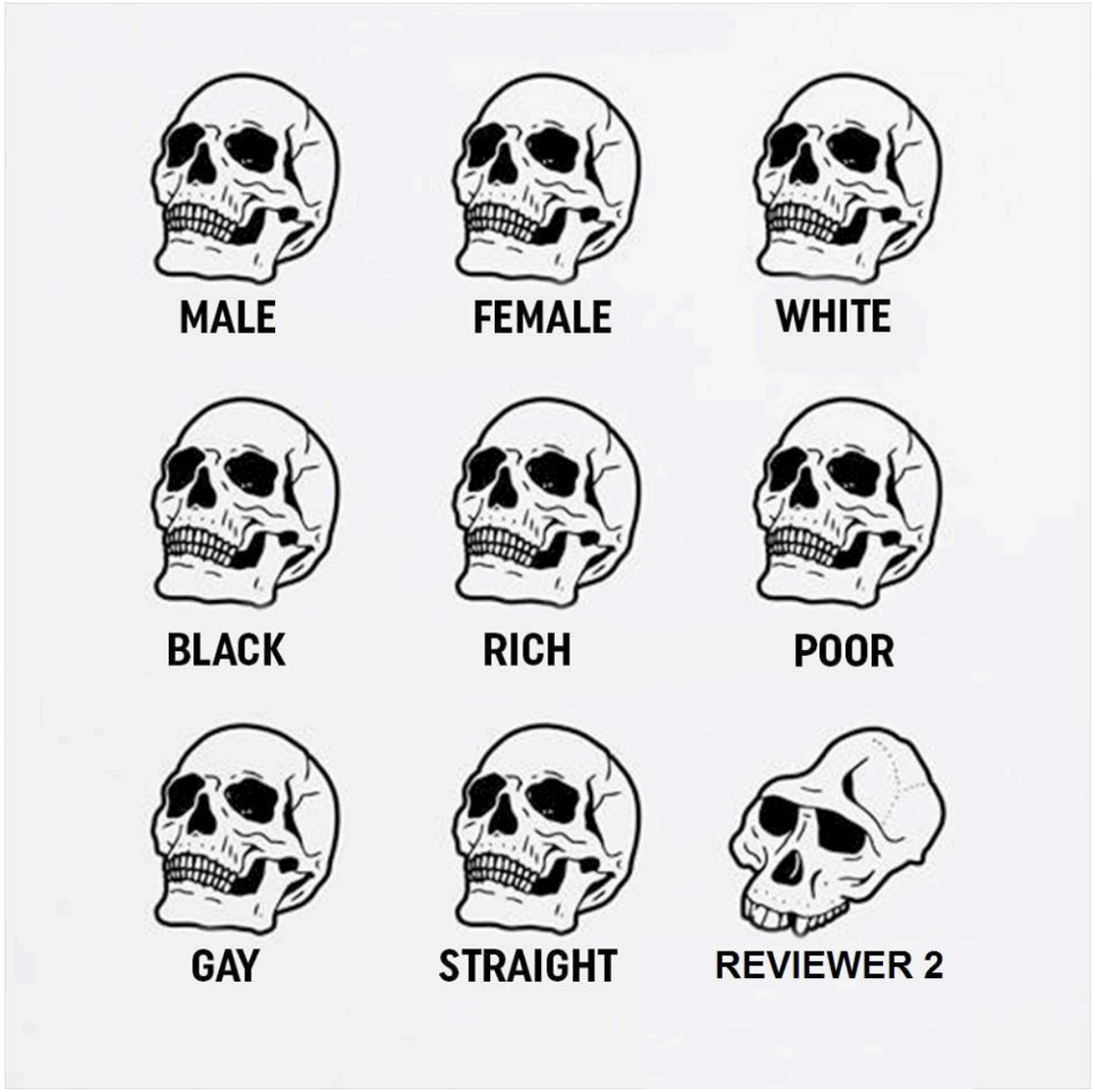
In the second part of this document, we introduce a higher-order probabilistic programming language, with a functionality analogous to that of established programming languages. This affords the opportunity to define models with dynamic computation graphs, at the cost of requiring inference methods that generate samples by repeatedly executing the program. Foundational inference algorithms for this kind of probabilistic programming language are explained in the context of an interface between program executions and an inference controller.

This document closes with a chapter on advanced topics which we believe to be, at the time of writing, interesting directions for probabilistic programming research; directions that point towards a tight integration with deep neural network research and the development of systems for next-generation artificial intelligence applications.

Extra : UBC CS Is Hiring

The Department of Computer Science at the University of British Columbia is inviting applications for **up to 4 positions** at the rank of **Assistant Professor**. We invite applications from all areas of computer science, with one of the primary areas of focus being in computer systems, broadly construed. **One position at a higher rank** could be considered for an individual with exceptional qualifications.


POSITIVE
SPACE


MALE
FEMALE
WHITE
BLACK
RICH
POOR
GAY
STRAIGHT
REVIEWER 2

Questions

- Do we ever want random variables in the middle of a computation?
- When will constant-time approximate inference algorithms fail?
- Which structure actually matters?
  - Does fine-grain deterministic structure matter?
  - Does course-grain (statistical) structure matter?
- If $\mathbf{q}$ is sufficiently (infinitely) flexible, what's the difference between learning with inference network and guide program objectives?