Towards Verified Stochastic Variational Inference for Probabilistic Programs

Wonyeol Lee, Hangyeol Yu, Xavier Rival and Hongseok Yang

June 2022

Probabilistic programming

Basic features:

- computation **over distributions**
- **sampling**
  i.e., draw a value from a distribution
- **conditioning** / **scoring**
  i.e., tune weight of executions based on observation

Advanced features: learning model parameters

Implementations: Edward, ProbTorch, Pyro, Stan,...

In this talk, **we consider Pyro** more specifically, applies to others too...

Variational inference

Problem

Given:

1. **a (potentially complex) model**
   description of a system / real data observations
   relies on sampling for, e.g., modeling / noisy measurement
2. **a (simpler) description**, referred to as **guide**
   sampling based on **unknown parameters**, but **with no observation**

Can we **infer optimal values of unknown parameters ?**

Example (Pyro):

- model describing a repeated coin tossing experiment
- guide describing a coin with biasedness given as parameter
- inference of the biasedness parameter based on experiment

Variational inference issues

Solution to the inference problem

several inference algorithms based on:

- collection of **families of executions**, with their **probability density**
- global **optimisation**, e.g., gradient descent

Pyro examples: include non trivial machine learning applications
e.g., variational auto-encoders
e.g., applications to basic MNIST number recognition

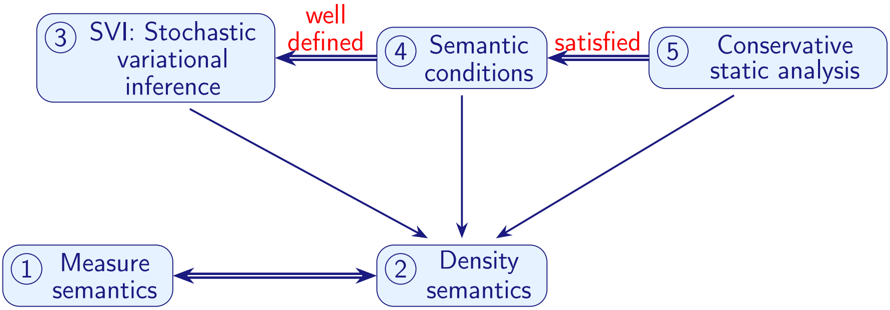
However:

Inference algorithms rely on non trivial theorems

- Are all the required assumptions always satisfied ?
- What happens otherwise ?

Our approach

To **address semantic definition issues**, we **follow a classical PL/static analysis approach:**



- standard semantics: in terms of measurable functions
- but models developed around **density** functions

Outline

A first, very basic model

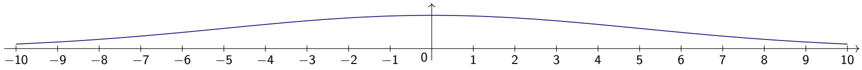
Model Pyro code:

```
def model():
  v = pyro.sample("v", Normal(0., 5.))
```

Meaning:

- `sample`: draws a value based on a distribution
  in this case, normal distribution, mean 0, standard deviation 5
- i.e., values of variable `v` distributed around 0
  with some imprecision

Distribution over executions based on the final value of `v`:



A second, more interesting model

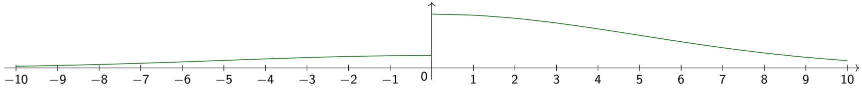
Model Pyro code:

```
def model():
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):
    pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:
    pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

Meaning:

- `sample` without `obs=...`: sampling, as before
- `sample` with `obs=...`: conditioning determined by observation

Distribution on v:



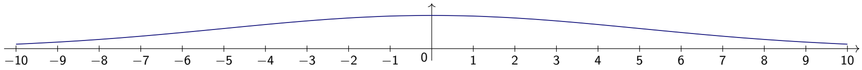
Distribution defined by the model

Model Pyro code:

```
def model():
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):
    pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:
    pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

Prior on v, before observation taken into account:

i.e., when observations on the value of obs are ignored

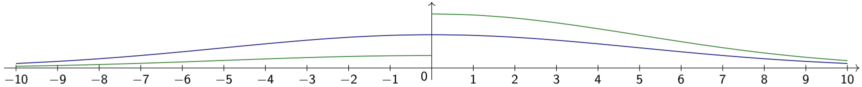


Distribution defined by the model

Model Pyro code:

```
def model():
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):
    pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:
    pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

Posterior distribution on v, after observations on obs
and compared with the prior:

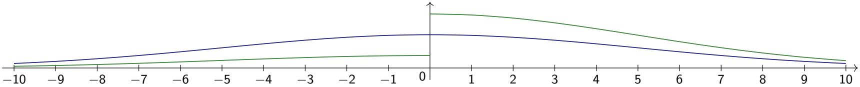


Distribution defined by the model

Model Pyro code:

```
def model():
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):
    pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:
    pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

Posterior distribution on v, after observations on obs
and compared with the prior:



Can we **discover a simpler, accurate enough approximation of the posterior ?**

Model approximation with a parameterized "guide"

Idea: specify a template for a family of candidate functions to approximate the posterior, then choose among them the most suitable one

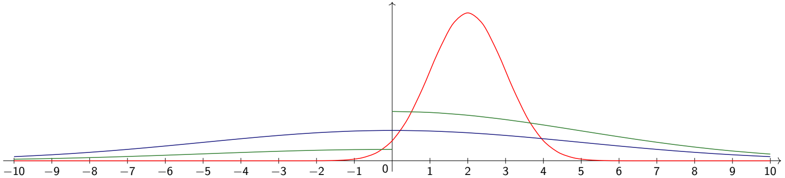
Guide

Companion program **with randomized parameter**, aimed at **approximating the posterior distribution** defined in the model

In our example: **sampling the parameter** from a **normal distribution**

```
def guide():
  theta = pyro.param("theta", 3.)
  v = pyro.sample("v", Normal(theta, 1.))
```

One instance of the guide, with a positive θ (**expected outcome**)



Inference: selection of a good parameter value

- Guide: specifies **a family of candidates model approximations** characterized by a parameter
- **Inference:** computes the *optimal* value of the parameter θ

Notion of optimality ? KL divergence (Kullback-Leibler)

Given two probability distributions p_0, p_1 over the same measurable set, their **KL divergence** writes down as:

$$\mathbf{D}_{\mathrm{KL}}(p_0, p_1) = \mathbb{E}_{p_0}\left(\log \frac{p_0}{p_1}\right) = \int \log \frac{dp_0}{dp_1} dp_0$$

defined **when p_0 absolutely continuous wrt p_1**, i.e.,
for all measurable x, $p_1(x) = 0 \Longrightarrow p_0(x) = 0$

- $\mathbf{D}_{\mathrm{KL}}(p_0, p_1) \geq 0$
- $\mathbf{D}_{\mathrm{KL}}(p_0, p_1) = 0$ if and only if p_0 and p_1 are equal almost everywhere

Inference principle

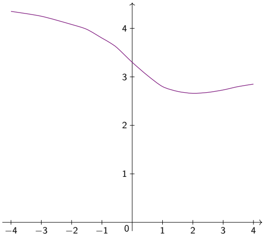
Inference goal

Compute an ideal value of θ, using an **optimization algorithm**

Application to the **inference problem**
two distributions over sampled variables $\mathtt{v}$:

- $p(\mathtt{v}, \mathtt{obs} = 0)$:
  posterior probability distribution over
  $\mathtt{v}$ defined by the model
  (with the observation $\mathtt{obs} = 0$)
- $q_\theta(\mathtt{v})$:
  guide probability distribution
  (parameterized by θ)

Plot of
$\mathbf{D_{KL}}(p(\mathtt{v}), q_\theta(\mathtt{v}, \mathtt{obs} = 0))$
as a function of θ:



Optimization objective: $\text{argmin}_\theta \mathbf{D_{KL}}(p(\mathtt{v}), q_\theta(\mathtt{v}, \mathtt{obs} = 0))$

Stochastic variational inference (SVI)

Principle:

- apply a **gradient descent algorithm** to KL divergence
  to compute optimal value of θ
- use **stochastic approximation of the gradient**
  i.e., generate samples based on current θ to estimate gradient

Algorithm to compute local minimum:

$$\begin{cases} \text{select } \theta_0 \\ \textbf{repeat } K \text{ times} \\ \quad \theta_{n+1} \leftarrow \theta_n - \lambda \overline{\nabla \mathbf{D}_{\mathrm{KL}}(p(\mathtt{v}), q_\theta(\mathtt{v}, \mathtt{obs} = 0))}_{\theta=\theta_n, N} \end{cases}$$

- λ: **learning rate**, typically small, e.g., $\lambda = 0.01$
- $\overline{\nabla \mathbf{D}_{\mathrm{KL}}(p(\mathtt{v}), q_\theta(\mathtt{v}, \mathtt{obs} = 0))}_{\theta=\theta_n, N}$: **gradient approximation over N samples**

Stochastic variational inference (SVI)

Principle:

- apply a **gradient descent algorithm** to KL divergence
  to compute optimal value of θ
- use **stochastic approximation of the gradient**
  i.e., generate samples based on current θ to estimate gradient

Algorithm to compute local minimum:

$$\begin{cases} \text{select } \theta_0 \\ \textbf{repeat } K \text{ times} \\ \quad \theta_{n+1} \leftarrow \theta_n - \lambda \overline{\nabla \mathbf{D}_{\mathrm{KL}}(p(\mathrm{v}), q_\theta(\mathrm{v}, \mathtt{obs} = 0))}_{\theta=\theta_n, N} \end{cases}$$

Pyro application of stochastic variational inference:

```
svi = SVI(model, guide, Adam({"lr": 1.0e-2}), loss=Trace_ELBO())
for step in range(2000):
  svi.step()
```

Stochastic variational inference (SVI)

Principle:

- apply a **gradient descent algorithm** to KL divergence
  to compute optimal value of θ
- use **stochastic approximation of the gradient**
  i.e., generate samples based on current θ to estimate gradient

Algorithm to compute local minimum:

$$\begin{cases} \text{select } \theta_0 \\ \textbf{repeat } K \text{ times} \\ \quad \theta_{n+1} \leftarrow \theta_n - \lambda \overline{\nabla \mathbf{D}_{\mathrm{KL}}(p(\mathtt{v}), q_\theta(\mathtt{v}, \mathtt{obs} = 0))}_{\theta=\theta_n, N} \end{cases}$$

Is it always guaranteed to work ?

Another model-guide pair

Excerpt from the **Pyro webpage examples**...

Model:

```
def model(...):
  ...
  sigma = pyro.sample("sigma", Uniform(0., 10.))
  ...
  pyro.sample("obs", Normal(..., sigma), obs=...)
```

Guide:

```
def guide(...):
  ...
  loc = pyro.param("sigma_loc", 1., constraint=constraints.positive)
  ...
  sigma = pyro.sample("sigma", Normal(loc, 0.05))
```

Issue: KL-divergence is undefined

- domain of `sigma` in the **model:** $[0, 10]$
- domain of `sigma` in the **guide:** $\mathbb{R}$

Issues possibly leading to undefinedness of KL-divergence

Absolute continuity requirement:

- definition of KL-divergence: $\mathbf{D}_{\mathrm{KL}}(q_\theta, p) = \int \log \frac{\mathrm{d}q_\theta}{\mathrm{d}p} \mathrm{d}q_\theta$
- absolute continuity requirement:
  model distribution p and guide distribution q should have **the same zero probability regions**
- domain in **model** $[0, 10]$, in **guide** $\mathbb{R}$
  leads to **the violation of absolute continuity assumption**
  e.g., and **KL divergence is undefined**

Anther possible issue: integrability

- $\log \frac{\mathrm{d}q_\theta}{\mathrm{d}p} \mathrm{d}q_\theta$ may not be integrable
  . . . even if absolute continuity holds

Our goal: define **semantics** to let **static analysis** provide guarantees

Informal overview of potential SVI issues

Several assumptions are necessary:

- **KL-divergence must be defined, not ∞:**
  otherwise: undefined optimization objective
- **KL-divergence must be differentiable:**
  otherwise: incorrect gradient descent
- **the stochastic estimate of $\nabla \mathrm{D}_{\mathbf{KL}}(q_\theta, p)$ should be well-defined, and unbiased:**
  otherwise: incorrect computation of gradient descent approximation

Practical consequences are difficult to troubleshoot, e.g.,

- crashes or divergence of the inference engine
- incoherent / invalid optimization results
  may be very difficult to even notice

Outline

A basic imperative probabilistic programming language

A few assumptions:

- imperative control structures (while language),
- real numbers (not floating point)
- only normal distributions
- countable set of random variables, represented with strings

Basic syntax:

$$
\begin{array}{lll}
E, B, S & \text{real, boolean, string expressions} & \\
C & ::= & \text{commands} \\
 & \mid & \mathrm{skip} \mid C_0; C_1 \mid x := E \\
 & \mid & \mathrm{if}\, B\, \{C\}\, \mathrm{else}\, \{C\} \mid \mathrm{while}\, B\, \{C\} \\
 & \mid & x := \mathrm{sample}_{\mathcal{N}}(S, E_0, E_1) \\
 & & \quad S: \text{ random variable, } E_0: \text{ mean, } E_1: \text{ standard dev.} \\
 & \mid & \mathrm{score}_{\mathcal{N}}(E_0, E_1, E_2) \\
 & & \quad E_0: \text{ observed value, } E_1: \text{ mean, } E_2: \text{ standard dev.}
\end{array}
$$

Measure semantics

A **state** $(m, r) \in \textsf{States}$ is a pair made of

- a **store:** $m \in \textsf{Mem} = [\textsf{Vars} \to \mathbb{R}]$ (finite set of program variables)
- a **random database:** $r \in \textsf{RDBs} = [K \to \mathbb{R}]$
  (where K **finite set of random variables drawn so far**)

Executions:

- have a **weight** (or **execution score**) in $\mathbb{R}^+$
  initially 1, then computed based on score statments
- r is initially empty
  then, sampled random values get added to r in sample statements
- may not terminate/crash

Semantics general form:

$$[\![C]\!]_{\mathrm{meas}} : \textsf{States} \to (\mathcal{P}(\textsf{States} \times \mathbb{R}^+) \to [0, 1])$$

i.e., maps **one input state** into a (sub)-probability **distribution** over **sets of (output state, weight) pairs**

Measure semantics: a few cases

$$[\![C]\!]_{\mathrm{meas}} : \quad \textbf{States} \to (\mathcal{P}(\textbf{States} \times \mathbb{R}^+) \to [0,1]) \\ \equiv (\textbf{Mem} \times \textbf{RDBs}) \to \mathcal{P}(\textbf{Mem} \times \textbf{RDBs} \times \mathbb{R}^+)) \to [0,1]$$

Assignment statement $x := E$

$$[\![x := E]\!]_{\mathrm{meas}}(m, r)(A) \triangleq \mathbb{1}_{[(m[x \mapsto [\![E]\!](m)], r, 1) \in A]}$$

- weight is not modified
- variable x is updated in the store
- note: expressions should not read random variables directly

Measure semantics: a few cases

$$[\![C]\!]_{\mathrm{meas}} : \quad \textbf{States} \to (\mathcal{P}(\textbf{States} \times \mathbb{R}^+) \to [0,1])$$
$$\equiv (\textbf{Mem} \times \textbf{RDBs}) \to \mathcal{P}(\textbf{Mem} \times \textbf{RDBs} \times \mathbb{R}^+)) \to [0,1]$$

Assignment statement $x := E$

Sample statement $\mathrm{sample}_{\mathcal{N}}(S, E_0, E_1)$

$$[\![x := \mathrm{sample}_{\mathcal{N}}(S, E_1, E_2)]\!]_{\mathrm{meas}}(m, r)(A) \triangleq$$
$$\mathbb{1}_{[[\![S]\!](m) \notin \mathrm{dom}(r)]} \cdot \mathbb{1}_{[[\![E_2]\!](m) \in \mathbb{R}^{+*}]}$$
$$\cdot \int \mathrm{d}v \left(\mathcal{N}(v; [\![E_1]\!](m), [\![E_2]\!](m)) \cdot \mathbb{1}_{[(m[x \mapsto v], r[[\![S]\!](m) \mapsto v], 1) \in A]}\right)$$

- crashes when sampling from a rand. var. not in the random database
- crashes when standard deviation is negative
- otherwise updates the states and rdb with the sample,
  integrate over the density of the sampled distribution

Measure semantics: a few cases

$$[\![C]\!]_{\mathrm{meas}} : \quad \textbf{States} \to (\mathcal{P}(\textbf{States} \times \mathbb{R}^{+}) \to [0,1])$$
$$\equiv (\textbf{Mem} \times \textbf{RDBs}) \to \mathcal{P}(\textbf{Mem} \times \textbf{RDBs} \times \mathbb{R}^{+})) \to [0,1]$$

Assignment statement $x := E$

Sample statement $\mathrm{sample}_{\mathcal{N}}(S, E_0, E_1)$

Score statement $\mathrm{score}_{\mathcal{N}}(E_0, E_1, E_2)$

$$[\![\mathrm{score}_{\mathcal{N}}(E_0, E_1, E_2)]\!]_{\mathrm{meas}}(m, r)(A) \triangleq \mathbb{1}_{[[\![E_2]\!](m) \in \mathbb{R}^{+*}]} \cdot \mathbb{1}_{[(m, r, \mathcal{N}([\![E_0]\!](m); [\![E_1]\!](m), [\![E_2]\!](m))) \in A]}$$

- crashes when standard deviation is negative
- otherwise state left unmodified
  score the density of the distribution for the observed value

Measure semantics: a few cases

$$[\![C]\!]_{\text{meas}} : \quad \textbf{States} \to (\mathcal{P}(\textbf{States} \times \mathbb{R}^+) \to [0,1]) \\ \equiv (\textbf{Mem} \times \textbf{RDBs}) \to \mathcal{P}(\textbf{Mem} \times \textbf{RDBs} \times \mathbb{R}^+)) \to [0,1]$$

Assignment statement $x := E$

Sample statement $\mathrm{sample}_{\mathcal{N}}(S, E_0, E_1)$

Score statement $\mathrm{score}_{\mathcal{N}}(E_0, E_1, E_2)$

$[\![C]\!]_{\text{meas}}$ is measurable
and defines a sub-probability kernel from **States** to **States** $\times \mathbb{R}^+$

Measure semantics: a basic example

- Prior: x close to 0
- Posterior: noisy observation that x is close to 1

$$C \triangleq \begin{cases} x := \mathrm{sample}(''a'', 0, 5); \\ \mathrm{score}(x, 3, 1); \end{cases}$$

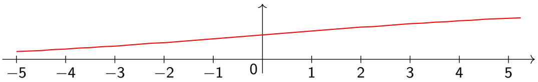
Measure semantics, starting from $m_I = \{x \mapsto ?\}$ and $r_I = \emptyset$,

$$\begin{aligned} &[\![C]\!]_{\mathrm{meas}}(m_I, r_I)(A) \\ &\quad = [\![C]\!]_{\mathrm{meas}}(m_I, r_I)(A \cap \{(\{x \mapsto v\}, \{a \mapsto v\}, \mathcal{N}(3; v, 1)) \mid v \in \mathbb{R}\}) \end{aligned}$$

(i.e., other ouptut state, density pairs do not count)

Cumulated measure, i.e., over $\{(\{x \mapsto v\}, \{a \mapsto v\}, \mathcal{N}(3; v, 1)) \mid v \leq \alpha\}$

$$\int_{-\infty}^{\alpha} [\![C]\!]_{\mathrm{meas}}(m_I, r_I)(\{(\{x \mapsto v\}, \{a \mapsto v\}, \mathcal{N}(3; v, 1)) \mid v \in \mathbb{R}\})\mathrm{d}v$$



From measure semantics to density semantics

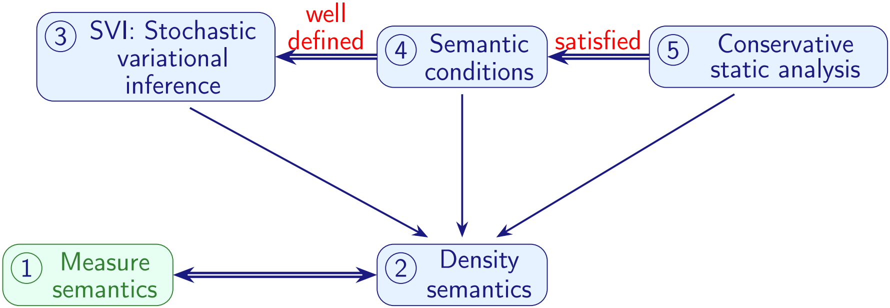
Posterior distribution over space of random databases:
starting from initial state $(m_I, \emptyset)$

$$\mathcal{M}(C, A) = \int [\![C]\!]_{\mathrm{meas}}(m_I, \emptyset)(\mathrm{d}(m, r, w))(w \cdot \mathbb{1}_{[(m,r) \in \mathbf{Mem} \times A]}),$$

From measure semantics to density semantics

Posterior distribution over space of random databases:
starting from initial state $(m_I, \emptyset)$

$$\mathcal{M}(C, A) = \int [\![C]\!]_{\mathrm{meas}}(m_I, \emptyset)(\mathrm{d}(m, r, w))(w \cdot \mathbb{1}_{[(m,r)\in \mathbf{Mem} \times A]}),$$



From measure semantics to density semantics

Posterior distribution over space of random databases:
starting from initial state $(m_I, \emptyset)$

$$\mathcal{M}(C, A) = \int [\![C]\!]_{\mathrm{meas}}(m_I, \emptyset)(\mathrm{d}(m, r, w))(w \cdot \mathbb{1}_{[(m,r)\in \mathbf{Mem}\times A]}),$$

Towards density semantics:

- should also include the density part in the semantics
  i.e., allow to write:

$$\mathcal{M}(C, A) = \int \rho(\mathrm{d}r)\,(\mathbb{1}_{[r\in A]} \cdot \mathbf{Dens}(C, m_I)(r))$$

  for some function **Dens** defined based **density semantics** $[\![\cdot]\!]_{\mathrm{dens}}$
- seeks for **operational flavor**
  i.e., easier to abstract for static analysis

Density semantics: a few cases

$$[\![C]\!]_{\text{dens}} : \textbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\} \to \textbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$
$$\equiv \textbf{Configs} \uplus \{\perp\} \to \textbf{Configs} \uplus \{\perp\}$$
$$\text{where} \quad \textbf{Configs} = (\textbf{Mem} \times \textbf{RDBs} \times \mathbb{R}^+ \times \mathbb{R}^+)$$

Configurations $(m, r, w, p) \in$ **Configs**:

- m: store
- r: random data-base

  each sample *pops* a value
- w: weight/score induced by observation
- p: probability induced by sampling

Density semantics: a few cases

$$\llbracket C \rrbracket_{\mathrm{dens}} : \mathbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\bot\} \to \mathbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\bot\}$$
$$\equiv \mathbf{Configs} \uplus \{\bot\} \to \mathbf{Configs} \uplus \{\bot\}$$
where $\quad \mathbf{Configs} = (\mathbf{Mem} \times \mathbf{RDBs} \times \mathbb{R}^+ \times \mathbb{R}^+)$

Assignment statement $x := E$

$$\llbracket x := E \rrbracket_{\mathrm{dens}}(m, r, w, p) \triangleq (m[x \mapsto \llbracket E \rrbracket(m)], r, 1, 1)$$

- weight and probability density not modified
- variable x is updated in the store

Density semantics: a few cases

$$\llbracket C \rrbracket_{\mathrm{dens}} : \textbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\bot\} \to \textbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\bot\}$$
$$\equiv \textbf{Configs} \uplus \{\bot\} \to \textbf{Configs} \uplus \{\bot\}$$
$$\text{where} \quad \textbf{Configs} = (\textbf{Mem} \times \textbf{RDBs} \times \mathbb{R}^+ \times \mathbb{R}^+)$$

Sample statement $\mathrm{sample}_{\mathcal{N}}(S, E_0, E_1)$

$$\llbracket x := \mathrm{sample}_{\mathcal{N}}(S, E_1, E_2) \rrbracket_{\mathrm{dens}}(m, r, w, p) \triangleq$$
$$\text{if } \llbracket S \rrbracket(m) \notin \mathrm{dom}(r) \vee \llbracket E_2 \rrbracket(m) \notin \mathbb{R}^{+*} \text{ then } \bot$$
$$\text{else } (m[x \mapsto r(\llbracket S \rrbracket(m))],\ r \setminus \llbracket S \rrbracket(m),\ w,\ p \cdot \mathcal{N}(r(\llbracket S \rrbracket(m)); \llbracket E_1 \rrbracket(m), \llbracket E_2 \rrbracket(r$$

- crashes when sampling from a random variables not in the random database or standard deviation is negative
- otherwise updates the states with the sample, score is unchanged
  probability density is multiplied by the density of the distribution

Density semantics: a few cases

$$[\![C]\!]_{\mathrm{dens}} : \textbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\bot\} \to \textbf{States} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\bot\}$$
$$\equiv \textbf{Configs} \uplus \{\bot\} \to \textbf{Configs} \uplus \{\bot\}$$
$$\text{where} \quad \textbf{Configs} = (\textbf{Mem} \times \textbf{RDBs} \times \mathbb{R}^+ \times \mathbb{R}^+)$$

Score statement $\mathrm{score}_{\mathcal{N}}(E_0, E_1, E_2)$

$$[\![\mathrm{score}_{\mathcal{N}}(E_0, E_1, E_2)]\!]_{\mathrm{dens}}(m, r, w, p) \triangleq$$
$$\text{if } ([\![E_2]\!](m) \notin \mathbb{R}^{+*}) \text{ then } \bot$$
$$\text{else } (m, r, w \cdot \mathcal{N}([\![E_0]\!](m); [\![E_1]\!](m), [\![E_2]\!](m)), p)$$

- crashes when standard deviation is negative
- otherwise state and probability density left unmodified
  score multiplied by the density of the observed distribution value

Density semantics: a basic example

Same program as in the previous example:

- Prior: x close to 0
- Posterior: noisy observation that x is close to 1

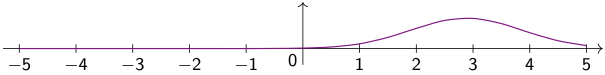
$$C \triangleq \begin{cases} x := \mathrm{sample}(''a'', 0, 5); \\ \mathrm{score}(x, 3, 1); \end{cases}$$

Semantics derived by simple calculation, starting from $m_I = \{x \mapsto ?\}$ and $r_I = \{a \mapsto v\}$,

$$[\![C]\!]_{\mathrm{dens}}((m_I, r_I), 1, 1) = ((\{x \mapsto v\}, \emptyset), \mathcal{N}(3; v, 1), \mathcal{N}(v; 0, 5))$$

Overall weighted density:

$$v \longmapsto \mathcal{N}(3; v, 1) \cdot \mathcal{N}(v; 0, 5)$$



Density semantics properties

Theorem: density definition

When $[\![C]\!]_{\mathrm{dens}}(m_I, r_I, 1, 1) = (m, \emptyset, w, p)$, we let:

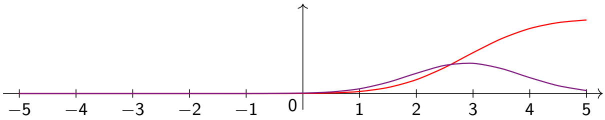
$$\mathbf{Dens}(C, m_I)(r_I) = w \cdot p$$

we have:

$$\begin{aligned}\mathcal{M}(C, A) &= \int \rho(\mathrm{d}r)\,(\mathbb{1}_{[r \in A]} \cdot \mathbf{Dens}(C, m_I)(r)) \\ &= \int [\![C]\!]_{\mathrm{meas}}(m_I, \emptyset)(\mathrm{d}(m, r, w))(w \cdot \mathbb{1}_{[(m,r) \in \mathbf{Mem} \times A]}),\end{aligned}$$

Example: weighted density

$$\alpha \longmapsto \mathcal{N}(3; \alpha, 1) \cdot \mathcal{N}(\alpha; 0, 5)$$



Outline

Towards a semantic definition of SVI

Given **model program** C, we defined:

$$\mathcal{M}(C, A) = \int \rho(\mathrm{d}r) \, (\mathbb{1}_{[r \in A]} \cdot \mathbf{Dens}(C, m_I)(r))$$

It can be **normalized** into a **probability measure** iff

$$\mathcal{M}(C, \mathbf{RDBs}) \in \mathbb{R}^{+*}$$

Objective of SVI:

- identify a family of programs D_θ as potential approximants of C
  1. with $\mathcal{M}(D_\theta, \mathbf{RDBs}) = 1$, which is ensured **if D_θ always terminates**
  2. with density 1, which is ensured **if no occurrence of** score
- compute optimal θ to minimize

$$\mathbf{D}_{\mathrm{KL}}(\mathbf{Dens}(D_\theta, m_I), \mathbf{Dens}(C, m_I))$$

where **Dens** defined by the density semantics

SVI algorithm

Gradient estimate based on one sample:

$$\mathbf{GrEst}_\theta(r) = (\nabla_\theta \log \mathbf{Dens}(D_\theta, m_I)) \cdot \log \frac{\mathbf{Dens}(D_\theta, m_I)}{\mathbf{Dens}(C, m_I)}$$

Fixed parameters:

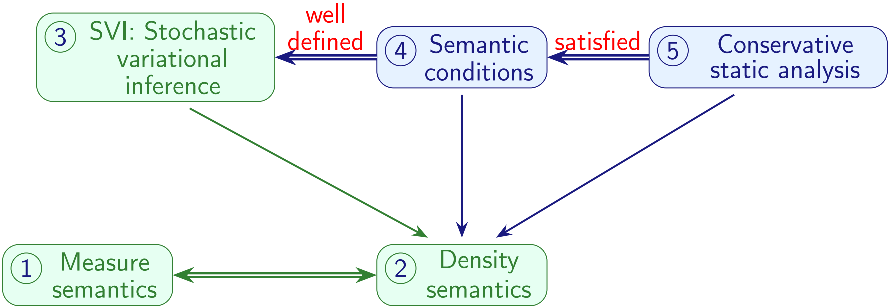
- N number of samples per iterate for stochastic estimation
- λ: learning rate, typically small, e.g., $\lambda = 0.01$

$$\begin{cases} \text{select } \theta_0 \\ \textbf{repeat } K \text{ times} \\ \qquad \text{sample } r_0, \ldots, r_{N-1} \\ \qquad \theta_{k+1} \leftarrow \theta_k - \lambda \cdot \frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta_k}(r_i) \end{cases}$$

Is $\frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta_k}(r_i)$ a good estimate of well-defined $\nabla_\theta \mathbf{D}_{\mathrm{KL}}(\mathbf{Dens}(D_\theta, m_I), \mathbf{Dens}(C, m_I))$???

SVI algorithms: conditions

Is $\frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta_k}(r_i)$ a **good estimate**
of **well-defined** $\nabla_\theta \mathrm{D}_{\mathrm{KL}}(\mathbf{Dens}(D_\theta, m_I), \mathbf{Dens}(C, m_I))$???



Unbiasedness

Theorem

If:

1. absolute continuity: $\mathbf{Dens}(D_\theta, m_I)(r) \Longrightarrow \mathbf{Dens}(C, m_I)(r)$
2. differentiability: $\theta \mapsto \mathbf{Dens}(D_\theta, m_I)(r)$ differentiable wrt all components
3. boundness of KL divergence
4. differentiability of KL divergence wrt all its arguments
5. integral permutation conditions on KL divergence and guide density $\int \nabla \ldots = \nabla \int \ldots$

Then:

$$\mathbb{E}(\nabla_\theta \mathbf{D}_{\mathrm{KL}}(\mathbf{Dens}(D_\theta, m_I), \mathbf{Dens}(C, m_I))) \equiv \frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta_k}(r_i)$$

Full version:

Towards Verified Stochastic Variational Inference for Probabilistic Programs
Wonyeol Lee, Hangyeol Yu, Xavier Rival and Hongseok Yang
POPL'20

Discharging requirements by restrictions + static analysis

A few syntactic restrictions over model/guide pairs:

- finitely many control flow branches
- fixed and finite set of random variables samples on each branch

Conservative static analysis applied to model/guide pair to **verify theorem hypotheses 1 and 2:**

- absolute continuity: $\mathbf{Dens}(D_\theta, m_I)(r) \Longrightarrow \mathbf{Dens}(C, m_I)(r)$
- differentiability: $\theta \mapsto \mathbf{Dens}(D_\theta, m_I)(r)$ differentiable wrt all components

Assumptions 3, 4, and 5 implied by stronger properties that could be verified by **static analysis**, by checking on other functions:

- boundness
- continuous differentiability

(analysis not done yet)

Outline

Abstract interpretation-based static analysis

Principles of static analysis

by **abstract interpretation**

1. start from a reference concrete semantics
   here: $[\![C]\!]_{\text{dens}} \in \mathcal{D}$
   where $\mathcal{D} = [\textbf{Configs} \uplus \{\bot\} \to \textbf{Configs} \uplus \{\bot\}]$
2. select **a family of abstract predicates** $\mathcal{D}^\sharp$
   with concretization function $\gamma : \mathcal{D}^\sharp \to \mathcal{D}$
   ideally with an efficient machine representation
3. derive a **computable, sound abstract semantics** $[\![C]\!]^\sharp$
   soundness:

$$[\![C]\!]_{\text{dens}} \in \gamma([\![C]\!]^\sharp)$$

Results are **sound:** accounts for all program behaviors
incomplete: spurious behaviors may be included
$\Rightarrow$ verification of correct programs may fail

A generic static analysis

We set up a static analysis, **parameterized by an abstract domain:**
Logical predicates + **representation** + **algorithms**

Abstract domain

An **abstract domain** comprises a set of abstract predicates $\mathcal{D}^\sharp$ and:

- **concretization function** $\gamma : \mathcal{D}^\sharp \to \mathcal{D}$
- **least element** $\bot$ with $\gamma(\bot) = \emptyset$
- **widening operator** $\mathit{widen}^\sharp : \mathcal{D}^\sharp \times \mathcal{D}^\sharp \longrightarrow \mathcal{D}^\sharp$
  over-approximating $\cup$
  and enforcing termination on all sequences of abstract iterates
- **abstract composition** $\mathit{comp}^\sharp : \mathcal{D}^\sharp \times \mathcal{D}^\sharp \longrightarrow \mathcal{D}^\sharp$
  soundness: $\forall g_0 \in \gamma(d_0^\sharp), \forall g_1 \in \gamma(d_1^\sharp),\ (g_0 \circ g_1) \in \gamma(\mathit{comp}^\sharp(d_0^\sharp, d_1^\sharp))$
- **abstract conditions, assignment, sample and score operations**
  satisfying similar soundness conditions

Static analysis construction and soundness

Definition of the analysis by induction over the syntax:

$$
\begin{array}{rcl}
[\![\mathrm{skip}]\!]^\sharp & \triangleq & \mathit{skip}^\sharp \\
[\![\mathrm{if}\ E\ \{C_0\}\ \mathrm{else}\ \{C_1\}]\!]^\sharp & \triangleq & \mathit{cond}^\sharp(E)([\![C_0]\!]^\sharp, [\![C_1]\!]^\sharp) \\
[\![x := E]\!]^\sharp & \triangleq & \mathit{assign}^\sharp(x, E) \\
[\![x := \mathrm{sample}_{\mathcal{N}}(S, E_1, E_2)]\!]^\sharp & \triangleq & \mathit{sample}^\sharp(x, S, E_1, E_2) \\
[\![C_0; C_1]\!]^\sharp & \triangleq & \mathit{comp}^\sharp([\![C_1]\!]^\sharp, [\![C_0]\!]^\sharp) \\
[\![\mathrm{score}_{\mathcal{N}}(E_0, E_1, E_2)]\!]^\sharp & \triangleq & \mathit{score}^\sharp(E_0, E_1, E_2) \\
[\![\mathrm{while}\ E\ \{C\}]\!]^\sharp & \triangleq & \mathbf{lfp}(\lambda d^\sharp \cdot \mathit{cond}^\sharp(E)(\mathit{comp}^\sharp(d^\sharp, [\![C]\!]^\sharp), \mathit{skip}^\sharp))
\end{array}
$$

Theorem: static analysis soundness

For all command C:

$$[\![C]\!]_{\mathrm{dens}} \in \gamma([\![C]\!]^\sharp)$$

$\Rightarrow$ next step: set up several instances of abstract domains

First instance: static analysis for support/guide match

Abstraction

We let: $\mathcal{D}^\sharp = \{\bot^\sharp, \top^\sharp\} \uplus \mathcal{P}(\textbf{String})$

and:

$$\begin{array}{rlcl} \gamma: & \bot^\sharp & \longmapsto & \lambda(m, r, w, p) \cdot \bot \\ & \top^\sharp & \longmapsto & \mathcal{D} \\ & S & \longmapsto & \{g \in \mathcal{D} \mid \forall(m, r, w, p),\ g(m, r, w, p) = (s', \emptyset, w', p') \\ & & & \qquad \Longrightarrow \mathrm{dom}(r) = S\} \end{array}$$

A few transfer functions:

$$\begin{array}{rcl} comp^\sharp(\bot^\sharp, d^\sharp) = comp^\sharp(d^\sharp, \bot^\sharp) & = & \bot^\sharp \\ comp^\sharp(\top^\sharp, d^\sharp) = comp^\sharp(d^\sharp, \top^\sharp) & = & \top^\sharp \\ comp^\sharp(S_0, S_1) & = & \begin{cases} S_0 \uplus S_1 & \text{if } S_0 \cap S_1 = \emptyset \\ \top^\sharp & \text{otherwise} \end{cases} \\ sample^\sharp(x, S, E_0, E_1) & = & \{S\} \\ score^\sharp(x, E_0, E_1, E_2) & = & \emptyset \end{array}$$

First instance: static analysis for support/guide match

Abstraction

We let: $\mathcal{D}^\sharp = \{\bot^\sharp, \top^\sharp\} \uplus \mathcal{P}(\mathbf{String})$

and:

$$\begin{array}{rlll} \gamma : & \bot^\sharp & \longmapsto & \lambda(m, r, w, p) \cdot \bot \\ & \top^\sharp & \longmapsto & \mathcal{D} \\ & S & \longmapsto & \{g \in \mathcal{D} \mid \forall(m, r, w, p),\ g(m, r, w, p) = (s', \emptyset, w', p') \\ & & & \qquad \Longrightarrow \mathrm{dom}(r) = S\} \end{array}$$

Example analysis:

```
def model():
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):
    pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:
    pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

Then: $[\![\mathtt{model}]\!]^\sharp = \{\mathtt{v}\}$

Second instance: static analysis for guide differentiability

Abstraction

We let

$$\mathcal{D}^\sharp = \mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}))$$

and $\gamma(X, Y, R)$ defined by:

- X: variables in X are definitely not modified
- Y: density is a $\mathcal{C}^1$ function of the variables in Y
- R: if $(V_0, V_1) \in R$ means the output value of the variables in V_1 is $\mathcal{C}^1$ in the variables in V_0

Other instances: **in the paper**

Outline

Static analysis support for basic language features

Our goal

Model/guide support correspondence analysis on **real Pyro programs**

Distributions:

- **not only normal distribution:** also uniform, beta, ...
- intuitively: a same rand. var. should be sampled **from the same distribution**, in both model and guide

Tensors:

- **multidimensional arrays:** $t : [1, N] \times [1, M] \times \ldots \times [1, P] \rightarrow \mathbb{R}$
- **basic:** operations on tensors of compatible dimensions
- **broadcasting:** operations on tensors of incompatible dimensions
- **plates:** grouping of tensor dimensions for optimization

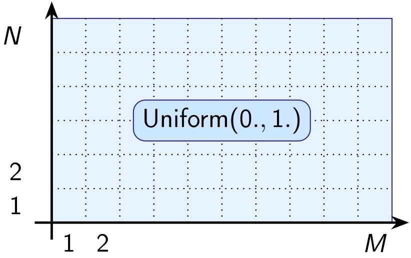
  should also be compatible in model and guide!

Abstraction for tensors and distributions

A model excerpt:

```
def model():
  for i in range (1,M):
    for j in range (1,N):
      ... = pyro.sample("x_{}_{}".format(i).format(j),
                                                  dist.Uniform(0,1))
      ...
```

Random-database: $\{\mathrm{x\_}i\mathrm{\_}j \mid 1 \leq i \leq M \wedge 1 \leq j \leq N\}$



Abstraction:

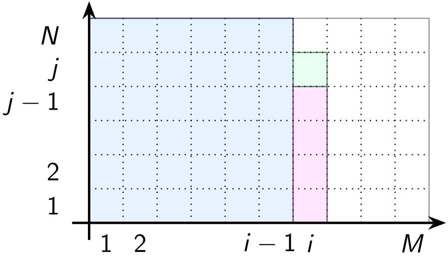
- should describe **zones** in multidimensional tensors
- should **bind distribution information** to zones

Abstraction for tensors and distributions

Precise tensor state abstraction:

$\mathcal{D}^\sharp$ elements:	finite set of (**tensor zone**, **distribution**) pairs
tensor zone:	tensor block $\cup \ldots \cup$ tensor block
tensor block:	range $\times \ldots \times$ range
range:	symbolic left bound $\times$ symbolic right bound
symbolic bound:	finite set of equal symbolic expression of prog. vars

Example invariant, after i, j iterations, one zone, **three tensor blocks:**



Evaluation: setup

Collection of programs using/configurable with `Trace_ELBO` estimator:

- 39 Pyro regression tests: small programs
- 8 Pyro examples: realistic probabilistic model implementations

Pyro example	size (LOCs)	sample dims	score dims	θ dims
br (Bayesian regression)	27	10	170	9
csis (inference compilation)	31	2	2	480
lda (amort. latent Dirich. alloc.)	76	21008	64000	121400
vae (variational autoencoder)	91	25600	200704	353600
sgdef (deep exponential family)	94	231280	1310720	231280
dmm (deep Markov model)	246	640000	281600	594000
ssvae (semi supervised vae)	349	24000	156800	844000
air (attend infer repeat)	410	20736	160000	6040859

Evaluation: results

On Pyro regression tests:

- 29 validated, 10 crashes of the analyser
- cause of crashes: **partial support** for plates, Pyro features

On Pyro examples:

Pyro example	valid ?	time (s)
br (Bayesian regression)	x	0.006
csis (inference compilation)	y	0.007
lda (amort. latent Dirich. alloc.)	x	0.014
vae (variational autoencoder)	y	0.005
sgdef (deep exponential family)	y	0.070
dmm (deep Markov model)	y	0.536
ssvae (semi supervised vae)	y	0.013
air (attend infer repeat)	y	4.093

- effectiveness:
  6 examples verified
  though complex
  code
- scalability:
  runtime under 1s
  for most tests
  one test takes 5s
  complex zones
- two issues found

Evaluation: discovered issues with model/guide mismatch

Bayesian regression (br):

- model: random variable `sigma` sampled from `Uniform(0,10)`
- guide: random variable `sigma` sampled from `Normal(...)`

Distribution support mismatch,
undefined SVI optimization objective

Amortized latent Dirichlet allocation (lda):

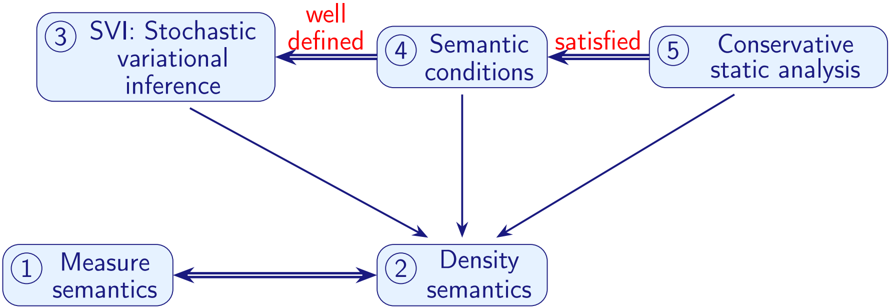
- model: random variable `doc_topics` sampled from `Dirichlet`, **continuous**
- guide: random variable `doc_topics` sampled from `Delta`, **discrete**

Distribution support mismatch, undefined SVI optim. objective,
but defined with other optim. engine *(Expectation Maximization)*

PL/Static analysis approach to provide guarantees on SVI

Encougaging results:

semantic formalization, semantic conditions, static analysis



Much remains to be done:

- more analyses to be implemented
- not full support for Python + Pytorch + Pyro features