

Stochastic Variational Inference

MPRI — Cours 2.40 “Probabilistic Programming Languages”

Xavier Rival

INRIA, ENS, CNRS

Nov, 20th. 2024

Information for the Exam

- No electronic device allowed
- Short personal note (2 pages max) allows

Probabilistic Programming Languages Inference

Inference

Evaluation of the posterior probability distribution

Caveats:

- exact inference either **impossible** or **not tractable** in general
- **many techniques** for approximate inference
several of them discussed in the previous classes

Today's plan: **stochastic variational inference (SVI)**

- intuition and definition
how to formalize the semantics of SVI
- issues: it may not work, **unless some conditions are satisfied**
questions: how to characterize these conditions, how to check them?

We consider **Pyro**, a probabilistic programming language implemented over Python with support for SVI, available at <https://pyro.ai>.

Formalization with a subset of the course language

Outline

- 1 Introduction
- 2 Basic Intuition Underlying SVI
- 3 Semantics for SVI
- 4 SVI
- 5 Ensuring correctness of SVI
- 6 Conclusion

Evaluation of probabilistic programs

**What does it mean to evaluate a probabilistic program ?
How to do that ?**

Enumeration:

- run all the executions and compute the probability of each of them
- only works in the finite case...

Importance sampling:

- compute a family of executions together with their probability
- executions are chosen at random (hopefully well)

Rejection sampling:

- compute a family of executions
- reject unlikely executions, accumulate those with high probability

Variational inference:

- search for a simpler program that is close enough
- ... more on this today...

A first, very basic model

We build a model step by step, to construct an example for SVI.

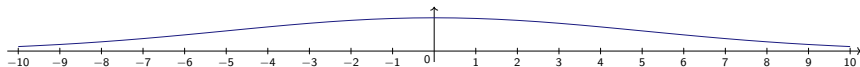
Model Pyro code:

```
def model():  
    x = 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 random v (python variable x) distributed around 0 with some imprecision

Distribution over executions based on the final value of x :



A second, more interesting model

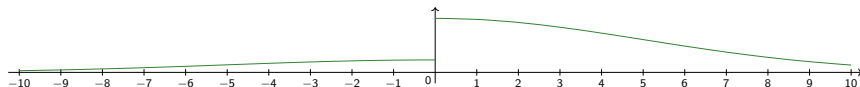
Model Pyro code:

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

Meaning:

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

Distribution on random variable v :



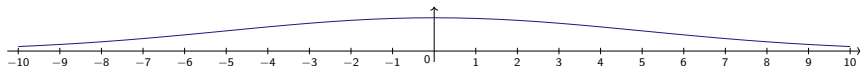
Distribution defined by the model

Model Pyro code:

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

Prior on random variable v , before observation taken into account:

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

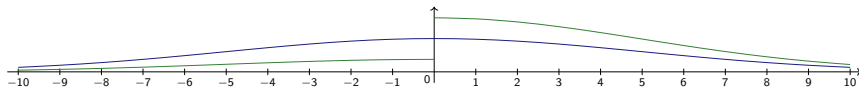


Distribution defined by the model

Model Pyro code:

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

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

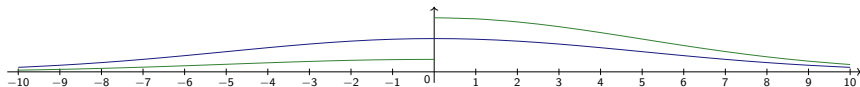


Distribution defined by the model

Model Pyro code:

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

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



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

Model approximation with a parameterized “guide”

SVI main Idea:

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

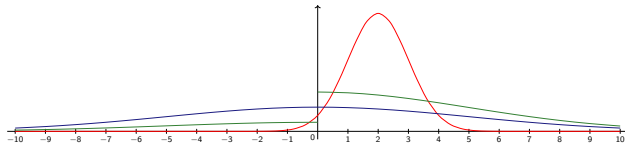
Guide

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

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



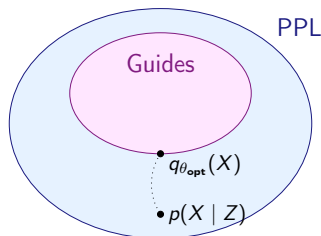
Objective of variational inference

SVI first question

How to define the optimization objective ? How to define the best guide ?

Data: probabilistic program P called **model**
samples variables X , observes variables Z

- We fix the family of guides programs with **special variables identified as parameters**
- Probabilistic programs denote **posterior probability distributions**
 - ▶ $p(X|Z)$ for model
 - ▶ $q_{\theta}(X)$ for guide instance θ
- We fix a **distance over distributions d** in general, KL divergence



Objective: guide parameter θ_{opt} instance that minimizes distance

Stochastic computation of an approximation of the objective

SVI second question

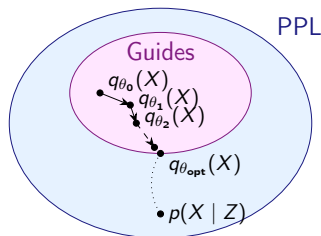
How to solve the optimization problem in practice ?

Gradient descent algorithm:

- 1 Pick initial parameter θ_0
- 2 **Estimate the gradient** ∇_{θ} of the distance at point θ_0 and make a step in that direction, to compute θ_1
- 3 Repeat to compute θ_2, \dots and **stop after K steps**

Stochastic approximation of the gradient at θ :

- Sample N runs, based on the guide
- Compute an estimate gradient based on the average variation



Convergence: based on properties of the optimization problem

Outline

- 1 Introduction
- 2 Basic Intuition Underlying SVI
- 3 Semantics for SVI
 - Measure semantics
 - Density semantics
- 4 SVI
- 5 Ensuring correctness of SVI
- 6 Conclusion

Towards a stochastic approximation of the distance

Main step of the gradient descent algorithm:

⇒ **estimation of the distance between distributions**

High-level overview of a single step of gradient descent (full algorithm shown later):

- 1 **sample** a number of executions (stochastic)
- 2 for each sampled execution, **compute probability density** (for short, density)
- 3 **produce a gradient approximation** based on the executions and their densities

How to compute the probability density of an expression ?

In the following, we propose a **density semantics** that answers the question:

- first, we recall the **kernel semantics**:
maps a state into a probability distribution over outcomes
- then, we construct a new semantics called **density semantics**, that produces not only output states but also probability density

Implementation: an evaluator following this density semantics (instrumented runtime)...

A basic imperative probabilistic programming language

We define a **minimalistic form** of BYOPPL.

A few assumptions:

- **imperative** control structures (while language),
- **real** numbers (not floating point)
- only normal distributions in the minimalistic syntax below though examples may use other distributions as well
- **countable set of random variables**, represented with **string** names

Basic syntax:

E, B, S real, boolean, string expressions

$C ::=$ commands

| **skip** | $C_0; C_1$ | $x := E$

| **if** $B \{C\}$ **else** $\{C\}$ | **while** $B \{C\}$

| $x := \text{sample}_{\mathcal{N}}(k, E_0, E_1)$

k : random variable name, E_0 : mean, E_1 : standard dev.

| **observe** $_{\mathcal{N}}(E_0, E_1, E_2)$

E_0 : observed value, E_1 : mean, E_2 : standard dev.

Towards two semantics

Common characteristics:

- executions **record all random choices**
random choices are **stored in a dictionary binding names to values**
- states comprise a **memory** and a **random dictionary**

Example:

$$x := \text{sample}_{\mathcal{N}}(s0, 0, 1);$$

$$y := \text{sample}_{\mathcal{N}}(s1, 2, 1);$$

initial state: $(s2 \mapsto 2), (x \mapsto 8, y \mapsto 9)$

final state: $(s0 \mapsto 0.5, s1 \mapsto 1.5, s2 \mapsto 2), (x \mapsto 0.5, y \mapsto 1.5)$

- consequence 1: a given random variable is sampled at most once
- consequence 2: initially present unsampled variables remain in the dictionary
- consequence 3: sampling from a variable already in the dictionary is blocking

Our two semantics:

- 1 **Kernel semantics:** maps states into probability distributions over states
- 2 **Density semantics:** maps states into states + probability density

Notations

We fix the following definitions:

- \mathbb{X} : set of **program variables**
- \mathbb{V} : set of **scalar values** (we assume real numbers, not floating point)
- $\mathbb{M} = [\mathbb{X} \rightarrow \mathbb{V}]$: set of **memory states**
(notation: $\mu \in \mathbb{M}$)
- \mathbb{K} : set of **random variables**, corresponding to strings
- $\mathbb{P} = [\mathbb{K} \rightarrow \mathbb{V}]$: set of **random databases** (or random dictionaries)
(notation: $\rho \in \mathbb{P}$)
- $\mathbb{S} = \mathbb{M} \times \mathbb{P}$: set of **states**
(notation: $\sigma \in \mathbb{S}$)
- given a measurable set \mathbb{A} , we note $\mathcal{M}(\mathbb{A})$ for the set of measurable subsets of \mathbb{A}

Assumptions

We fix the following assumptions:

- usual structure of **measurable space structures** over \mathbb{M} , \mathbb{P} , and \mathbb{S}
- notion of **probability kernel** over \mathbb{A} and measurable set \mathbb{A}' :
function from \mathbb{A} to probability distributions over \mathbb{A}' (previous lecture)
 - ▶ probability kernel: total measure is 1, noted $\mathcal{K}(\mathbb{A}, \mathbb{A}')$
 - ▶ sub-probability kernel: total measure ≤ 1 , noted $\mathcal{K}(\mathbb{A}, \mathbb{A}')_{\text{sub}}$

Semantics:

- we assume a semantics of expressions:
for all express E :

$$\llbracket E \rrbracket : \mathbb{M} \rightarrow \mathbb{V}$$

(random variables not used)

- Semantics of commands:

$$\llbracket C \rrbracket_{\mathcal{M}} \in \mathcal{K}(\mathbb{S}, \mathbb{S} \times \mathbb{R}^+)_{\text{sub}}$$

Or equivalently:

$$\llbracket C \rrbracket_{\mathcal{M}} : \mathbb{S} \rightarrow (\mathcal{M}(\mathbb{S} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]) \equiv (\mathbb{M} \times \mathbb{P}) \rightarrow \mathcal{M}(\mathbb{M} \times \mathbb{P} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]$$

Measure semantics: signature and a few cases

$$\llbracket C \rrbracket_{\mathcal{M}} : \mathbb{S} \rightarrow (\mathcal{M}(\mathbb{S} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]) \equiv (\mathbb{M} \times \mathbb{P}) \rightarrow \mathcal{M}(\mathbb{M} \times \mathbb{P} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]$$

Assignment statement $x := E$

$$\llbracket x := E \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \triangleq \mathbb{1}_{[(\mu[x \mapsto \llbracket E \rrbracket(\mu)], \rho), 1] \in S]}$$

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

Measure semantics: signature and a few cases

$$\llbracket C \rrbracket_{\mathcal{M}} : \mathbb{S} \rightarrow (\mathcal{M}(\mathbb{S} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]) \equiv (\mathbb{M} \times \mathbb{P}) \rightarrow \mathcal{M}(\mathbb{M} \times \mathbb{P} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]$$

Assignment statement $x := E$

Sample statement $x := \text{sample}_{\mathcal{N}}(s, E_0, E_1)$

$$\begin{aligned} \llbracket x := \text{sample}_{\mathcal{N}}(k, E_1, E_2) \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \triangleq & \\ & \mathbb{1}_{[k \notin \text{Dom}(\rho)]} \cdot \mathbb{1}_{[\llbracket E_2 \rrbracket(\mu) \in \mathbb{R}^{+*}]} \\ & \cdot \int \mathrm{d}v \left(\text{pdf}_{\mathcal{N}}(v; \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu)) \cdot \mathbb{1}_{[(\mu[x \mapsto v], \rho[k \mapsto v], 1) \in S]} \right) \end{aligned}$$

- 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 new sample,
integrate over the density of the sampled distribution
and do this for all possible samples (hence the sum)

Notation: $\text{pdf}_{\mathcal{N}}(v; m, d)$: **probability density** at v of the normal distribution of mean m and standard deviation s

Measure semantics: signature and a few cases

$$\llbracket C \rrbracket_{\mathcal{M}} : \mathbb{S} \rightarrow (\mathcal{M}(\mathbb{S} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]) \equiv (\mathbb{M} \times \mathbb{P}) \rightarrow \mathcal{M}(\mathbb{M} \times \mathbb{P} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]$$

Assignment statement $x := E$

Sample statement $x := \text{sample}_{\mathcal{N}}(s, E_0, E_1)$

Score statement $\text{observe}_{\mathcal{N}}(E_0, E_1, E_2)$

$$\begin{aligned} \llbracket \text{observe}_{\mathcal{N}}(E_0, E_1, E_2) \rrbracket_{\mathcal{M}}(\mu, \rho)(S) &\triangleq \\ &\mathbb{1}_{[\llbracket E_2 \rrbracket(\mu) \in \mathbb{R}^{+*}]} \cdot \mathbb{1}_{[(\mu, \rho), \text{pdf}_{\mathcal{N}}(\llbracket E_0 \rrbracket(\mu); \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu))] \in S} \end{aligned}$$

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

Measure semantics: signature and a few cases

$$\llbracket C \rrbracket_{\mathcal{M}} : \mathbb{S} \rightarrow (\mathcal{M}(\mathbb{S} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]) \equiv (\mathbb{M} \times \mathbb{P}) \rightarrow \mathcal{M}(\mathbb{M} \times \mathbb{P} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1]$$

Assignment statement $x := E$

Sample statement $x := \text{sample}_{\mathcal{N}}(s, E_0, E_1)$

Score statement $\text{observe}_{\mathcal{N}}(E_0, E_1, E_2)$

For all command C , $\llbracket C \rrbracket_{\mathcal{M}}$ is measurable
and defines a sub-probability kernel from \mathbb{S} to $\mathbb{S} \times \mathbb{R}^+$

Measure semantics (or kernel semantics)

$$\begin{aligned}
& \llbracket \text{skip} \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \quad \triangleq \mathbb{1}_{[(\mu, \rho), 1] \in S} \\
& \llbracket x := E \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \quad \triangleq \mathbb{1}_{[(\mu[x \mapsto \llbracket E \rrbracket(\mu)], \rho), 1] \in S} \\
& \llbracket C_0; C_1 \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \quad \triangleq \int \llbracket C_0 \rrbracket_{\mathcal{M}}(\mu, \rho)(d(\sigma_0, w_0)) \int \llbracket C_1 \rrbracket_{\mathcal{M}}(\sigma_0)(d(\sigma_1, w_1)) \mathbb{1}_{[(\sigma_1, w_0 w_1) \in S]} \\
& \llbracket \text{if } B \{ C_0 \} \text{else} \{ C_1 \} \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \quad \triangleq \mathbb{1}_{[\llbracket B \rrbracket(\mu) = \text{true}]} \cdot \llbracket C_0 \rrbracket_{\mathcal{M}}(\mu, \rho)(S) + \mathbb{1}_{[\llbracket B \rrbracket(\mu) = \text{false}]} \cdot \llbracket C_1 \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \llbracket \text{while } B \{ C \} \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \quad \triangleq (\text{Fix } F)(\mu, \rho)(S) \\
& \quad \text{where } F(\phi)(\mu, \rho)(S) = \mathbb{1}_{[\llbracket B \rrbracket(\mu) = \text{false}]} \cdot \mathbb{1}_{[(\mu, \rho), 1] \in S} \\
& \quad \quad + \mathbb{1}_{[\llbracket B \rrbracket(\mu) = \text{true}]} \cdot \int \llbracket C \rrbracket_{\mathcal{M}}(\mu, \rho)(d(\sigma_0, w_0)) \int \phi(\sigma_0)(d(\sigma_1, w_1)) \mathbb{1}_{[(\sigma_1, w_0 w_1) \in S]} \\
& \llbracket x := \text{sample}_{\mathcal{N}}(k, E_1, E_2) \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \quad \triangleq \mathbb{1}_{[k \notin \text{Dom}(\rho)]} \cdot \mathbb{1}_{[\llbracket E_2 \rrbracket(\mu) \in \mathbb{R}^{+*}]} \\
& \quad \quad \cdot \int d\nu (\text{pdf}_{\mathcal{N}}(\nu; \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu)) \cdot \mathbb{1}_{[(\mu[x \mapsto \nu], \rho[k \mapsto \nu], 1) \in S]}) \\
& \llbracket \text{observe}_{\mathcal{N}}(E_0, E_1, E_2) \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \\
& \quad \triangleq \mathbb{1}_{[\llbracket E_2 \rrbracket(\mu) \in \mathbb{R}^{+*}]} \cdot \mathbb{1}_{[(\mu, \rho), \text{pdf}_{\mathcal{N}}(\llbracket E_0 \rrbracket(\mu); \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu))] \in S]}
\end{aligned}$$

A basic example

We consider the program: $C \triangleq \begin{cases} x := \text{sample}(a, 0, 5); \\ \text{observe}(x, 3, 1); \end{cases}$

- prior: x close to 0, low confidence
- posterior: noisy observation that x is close to 3

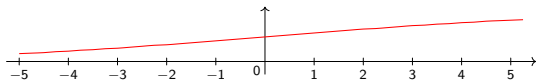
Measure semantics, starting from $\mu_I = \{x \mapsto ?\}$ and $\rho_I = \emptyset$,

$$\llbracket C \rrbracket_{\mathcal{M}}(\mu_I, \rho_I)(S) = \llbracket C \rrbracket_{\mathcal{M}}(\mu_I, \rho_I)(S \cap \{(\{x \mapsto v\}, \{a \mapsto v\}, \text{pdf}_{\mathcal{N}}(3; v, 1)) \mid v \in \mathbb{R}\})$$

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

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

$$\int_{-\infty}^{\alpha} \llbracket C \rrbracket_{\mathcal{M}}(\mu_I, \rho_I)(\{(\{x \mapsto v\}, \{a \mapsto v\}, \text{pdf}_{\mathcal{N}}(3; v, 1)) \mid v \in \mathbb{R}\}) dv$$



Issue: per execution weight and probability over executions remain separate

Outline

- 1 Introduction
- 2 Basic Intuition Underlying SVI
- 3 Semantics for SVI
 - Measure semantics
 - Density semantics
- 4 SVI
- 5 Ensuring correctness of SVI
- 6 Conclusion

Towards a density semantics

For the **definition of SVI**, the measure semantics has several **limitations**:

- the weight of executions (**observe** commands) and the measure over output configurations are computed separately
- as mentioned earlier, we look for a way to compute the probability density of each execution together with its output

Important notes:

- as we seek for a semantics that maps an initial state to an output configuration, we need to assume random samples fixed beforehand and are **consumed** during the execution
i.e., the ρ input collects values to be sampled, each **sample** pops a value
note this is the converse of the measure semantics convention
- we also expect this semantics to be **deterministic**, *i.e.*, for any given initial state, the execution of a command should produce a single output configuration
- executions **may not terminate or may crash** so a special output configuration is needed
i.e., \perp denotes executions that either fail or do not terminate

Density semantics basic definition

We recall the form of the measure semantics:

$$\llbracket C \rrbracket_{\mathcal{M}} : \mathbb{S} \rightarrow (\mathcal{M}(\mathbb{S} \times \mathbb{R}^+) \rightarrow_{\mathcal{M}} [0, 1])$$

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

When $\llbracket C \rrbracket_{\mathcal{D}}(\mu, \rho) = (\mu', \rho', w', p')$:

- μ' denotes the new memory
- ρ' denotes the remaining part of the random dictionary
- w' denotes the execution weight (scoring)
- p' denotes its probability density

Definition of the semantics: exercise!

Density semantics of the skip command

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

Command C :

skip

Measure semantics:

$$\llbracket \text{skip} \rrbracket_{\mathcal{M}}(\mu, \rho)(S) = \mathbb{1}_{[(\mu, \rho), 1) \in S]}$$

Exercise

1 Propose a definition for $\llbracket \cdot \rrbracket_{\mathcal{D}}$

Density semantics of assignment commands

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

Command C :

$$x := E$$

Measure semantics:

$$\llbracket x := E \rrbracket_{\mathcal{M}}(\mu, \rho)(S) = \mathbb{1}_{[(\mu[x \mapsto \llbracket E \rrbracket(\mu)], \rho), 1] \in S}$$

Exercise

- 1 **Propose a definition for $\llbracket . \rrbracket_{\mathcal{D}}$**
under the assumption that expressions never crash
- 2 What would happen if we assume expressions may crash

Density semantics of sequences

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

Command C :

$$C_0; C_1$$

Measure semantics:

$$\begin{aligned} \llbracket C_0; C_1 \rrbracket_{\mathcal{M}}(\mu, \rho)(S) = \\ \int \llbracket C_0 \rrbracket_{\mathcal{M}}(\mu, \rho)(d(\sigma_0, w_0)) \int \llbracket C_1 \rrbracket_{\mathcal{M}}(\sigma_0)(d(\sigma_1, w_1)) \mathbb{1}_{[(\sigma_1, w_0 w_1) \in S]} \end{aligned}$$

Exercise

❶ **Propose a definition for $\llbracket \cdot \rrbracket_{\mathcal{D}}$**

❷ What happens with composition ?

Try to imagine a “lift” operator that makes $\llbracket \cdot \rrbracket_{\mathcal{D}}$ easier to compose and update the definition for $\llbracket \cdot \rrbracket_{\mathcal{D}}$ accordingly

Density semantics of condition tests

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

Command C :

$$\text{if } B \{ C_0 \} \text{else} \{ C_1 \}$$

Measure semantics:

$$\begin{aligned} \llbracket \text{if } B \{ C_0 \} \text{else} \{ C_1 \} \rrbracket_{\mathcal{M}}(\mu, \rho)(S) = \\ \mathbb{1}_{\llbracket B \rrbracket(\mu) = \text{true}} \cdot \llbracket C_0 \rrbracket_{\mathcal{M}}(\mu, \rho)(S) + \mathbb{1}_{\llbracket B \rrbracket(\mu) = \text{false}} \cdot \llbracket C_1 \rrbracket_{\mathcal{M}}(\mu, \rho)(S) \end{aligned}$$

Exercise

1 Propose a definition for $\llbracket \cdot \rrbracket_{\mathcal{D}}$

Density semantics of loop commands

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

Command C :

$\text{while } B \{ C \}$

Measure semantics:

$$\llbracket \text{while } B \{ C \} \rrbracket_{\mathcal{M}}(\mu, \rho)(S) = (\text{Fix } F)(\mu, \rho)(S)$$

where

$$F(\phi)(\mu, \rho)(S) = \mathbb{1}_{\llbracket B \rrbracket(\mu) = \text{false}} \cdot \mathbb{1}_{[(\mu, \rho), 1] \in S} \\ + \mathbb{1}_{\llbracket B \rrbracket(\mu) = \text{true}} \cdot \int \llbracket C \rrbracket_{\mathcal{M}}(\mu, \rho)(d(\sigma_0, w_0)) \int \phi(\sigma_0)(d(\sigma_1, w_1)) \mathbb{1}_{[(\sigma_1, w_0 w_1) \in S]}$$

Exercise

- 1 **Propose a definition for $\llbracket \cdot \rrbracket_{\mathcal{D}}$**
- 2 **Explain the assumptions required for the definition**

Density semantics of sample commands

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

Command C :

$$x := \text{sample}_{\mathcal{N}}(k, E_1, E_2)$$

Measure semantics:

$$\begin{aligned} \llbracket x := \text{sample}_{\mathcal{N}}(k, E_1, E_2) \rrbracket_{\mathcal{M}}(\mu, \rho)(S) = \\ \mathbb{1}_{[k \notin \text{Dom}(\rho)]} \cdot \mathbb{1}_{[\llbracket E_2 \rrbracket(\mu) \in \mathbb{R}^{+*}]} \\ \cdot \int d\nu \left(\text{pdf}_{\mathcal{N}}(\nu; \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu)) \cdot \mathbb{1}_{[(\mu[x \mapsto \nu], \rho[k \mapsto \nu], 1) \in S]} \right) \end{aligned}$$

Exercise

- 1 **Propose a definition for $\llbracket \cdot \rrbracket_{\mathcal{D}}$**
- 2 **Comment on error cases**

Density semantics of observe commands

Signature of the density semantics

$$\llbracket C \rrbracket_{\mathcal{D}} : \mathbb{S} \rightarrow_{\mathcal{M}} \mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$$

Command C :

$$\text{observe}_{\mathcal{N}}(E_0, E_1, E_2)$$

Measure semantics:

$$\begin{aligned} \llbracket \text{observe}_{\mathcal{N}}(E_0, E_1, E_2) \rrbracket_{\mathcal{M}}(\mu, \rho)(S) = \\ \mathbb{1}[\llbracket E_2 \rrbracket(\mu) \in \mathbb{R}^{+*}] \cdot \mathbb{1}[(\mu, \rho, \text{pdf}_{\mathcal{N}}(\llbracket E_0 \rrbracket(\mu); \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu))) \in S] \end{aligned}$$

Exercise

- 1 **Propose a definition for $\llbracket \cdot \rrbracket_{\mathcal{D}}$**
- 2 **Comment on error cases**

Density semantics

Lifting of semantic function $g : \mathbb{M} \times \mathbb{P} \rightarrow_{\mathcal{M}} \mathbb{M} \times \mathbb{P} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\}$, for composition:

$$\begin{aligned} g^\bullet(\perp) &= \perp \\ g^\bullet(\mu, \rho, w, p) &= \begin{cases} \perp & \text{if } g(\mu, \rho) = \perp \\ (\mu', \rho', w \cdot w', p \cdot p') & \text{if } g(\mu, \rho) = (\mu', \rho', w', p') \end{cases} \end{aligned}$$

Semantics:

$$\begin{aligned} \llbracket \text{skip} \rrbracket_{\mathcal{D}}(\mu, \rho) &= (\mu, \rho, 1, 1) \\ \llbracket x := E \rrbracket_{\mathcal{D}}(\mu, \rho) &= (\mu[x \mapsto \llbracket E \rrbracket(\mu)], \rho, 1, 1) \\ \llbracket C_0; C_1 \rrbracket_{\mathcal{D}}(\mu, \rho) &= \llbracket C_0 \rrbracket_{\mathcal{D}}^\bullet \circ \llbracket C_1 \rrbracket_{\mathcal{D}}(\mu, \rho) \\ \llbracket \text{if } B \{ C_0 \} \text{else} \{ C_1 \} \rrbracket_{\mathcal{D}}(\mu, \rho) &= \text{if } \llbracket B \rrbracket(\mu) = \text{true then } \llbracket C_0 \rrbracket_{\mathcal{D}}(\mu, \rho) \text{ else } \llbracket C_1 \rrbracket_{\mathcal{D}}(\mu, \rho) \\ \llbracket \text{while } B \{ C \} \rrbracket_{\mathcal{D}}(\mu, \rho) &= (\text{Fix } G)(\mu, \rho) \\ \text{where } G(\psi)(\mu, \rho) &= \text{if } \llbracket B \rrbracket(\mu) = \text{false then } (\mu, \rho, 1, 1) \text{ else } \psi^\bullet \circ \llbracket C \rrbracket_{\mathcal{D}}(\mu, \rho) \end{aligned}$$

$$\begin{aligned} \llbracket x := \text{sample}_{\mathcal{N}}(k, E_1, E_2) \rrbracket_{\mathcal{D}}(\mu, \rho) \\ = \begin{cases} \text{if } k \notin \text{Dom}(\rho) \vee \llbracket E_2 \rrbracket(\mu) \notin \mathbb{R}^{+*} \text{ then } \perp \\ \text{else } (\mu[x \mapsto \rho(k)], \rho \setminus \{k\}, 1, \text{pdf}_{\mathcal{N}}(\rho(k); \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu))) \end{cases} \end{aligned}$$

$$\begin{aligned} \llbracket \text{observe}_{\mathcal{N}}(E_0, E_1, E_2) \rrbracket_{\mathcal{D}}(\mu, \rho) \\ = \begin{cases} \text{if } (\llbracket E_2 \rrbracket(\mu) \notin \mathbb{R}^{+*}) \text{ then } \perp \\ \text{else } (\mu, \rho, \text{pdf}_{\mathcal{N}}(\llbracket E_0 \rrbracket(\mu); \llbracket E_1 \rrbracket(\mu), \llbracket E_2 \rrbracket(\mu)), 1) \end{cases} \end{aligned}$$

Density of an execution and example

Definition: execution probability density

When $\llbracket C \rrbracket_{\mathcal{D}}(\mu_I, \rho_I) = (\mu, \emptyset, w, p)$, we let:

$$\mathcal{D}\llbracket C \rrbracket(\mu_I, \rho_I) = w \cdot p$$

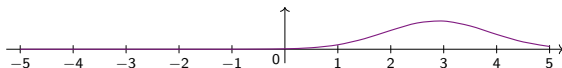
We consider again the program: $C \triangleq \begin{cases} x := \text{sample}(a, 0, 5); \\ \text{observe}(x, 3, 1); \end{cases}$

- prior: x close to 0, low confidence
- posterior: noisy observation that x is close to 3

Semantics derived by simple calculation, starting from $\mu_I = \{x \mapsto ?\}$ and $\rho_I(v) = \{a \mapsto v\}$,

$$\llbracket C \rrbracket_{\mathcal{D}}((\mu_I, \rho_I), 1, 1) = ((\{x \mapsto v\}, \emptyset), \text{pdf}_{\mathcal{N}}(3; v, 1), \text{pdf}_{\mathcal{N}}(v; 0, 5))$$

Overall weighted density: $v \mapsto \mathcal{D}\llbracket C \rrbracket(\mu_i, \rho_i(v)) = \text{pdf}_{\mathcal{N}}(3; v, 1) \cdot \text{pdf}_{\mathcal{N}}(v; 0, 5)$



Density of an execution and example

Definition: execution probability density

When $\llbracket C \rrbracket_{\mathcal{D}}(\mu_I, \rho_I) = (\mu, \emptyset, w, p)$, we let:

$$\mathcal{D}[\llbracket C \rrbracket](\mu_I, \rho_I) = w \cdot p$$

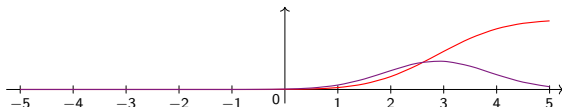
We consider again the program: $C \triangleq \begin{cases} x := \text{sample}(a, 0, 5); \\ \text{observe}(x, 3, 1); \end{cases}$

- prior: x close to 0, low confidence
- posterior: noisy observation that x is close to 3

Semantics derived by simple calculation, starting from $\mu_I = \{x \mapsto ?\}$ and $\rho_I(v) = \{a \mapsto v\}$,

$$\llbracket C \rrbracket_{\mathcal{D}}((\mu_I, \rho_I), 1, 1) = ((\{x \mapsto v\}, \emptyset), \text{pdf}_{\mathcal{N}}(3; v, 1), \text{pdf}_{\mathcal{N}}(v; 0, 5))$$

Cumulated weighted density: $v \mapsto \int_{-\infty}^v \mathcal{D}[\llbracket C \rrbracket](\mu_i, \rho_i(x)) dx$



Link between density semantics and measure semantics

Definition of density: when $\llbracket C \rrbracket_{\mathcal{D}}(\mu_I, \rho_I) = (\mu, \emptyset, w, p)$, we have $\mathcal{D}\llbracket C \rrbracket(\mu_I, \rho_I) = w \cdot p$

Theorem

Given a set of random dictionaries $P \subset \mathbb{P}$, we can measure the probability to evaluate exactly a $\rho \in P$ with either semantics, in an equivalent manner:

$$\begin{aligned} \mathcal{M}(C, P) &\triangleq \int (\mathbb{1}_{[\rho \in P]} \cdot |\mathrm{d}\rho| \cdot \mathcal{D}\llbracket C \rrbracket(\mu_I, \rho)) \\ &= \int \llbracket C \rrbracket_{\mathcal{M}}(\mu_I, \emptyset)(\mathrm{d}(\mu, \rho, w)) \cdot (w \cdot \mathbb{1}_{[\rho \in P]}) \end{aligned}$$

Proof: exercise!

Outline

- 1 Introduction
- 2 Basic Intuition Underlying SVI
- 3 Semantics for SVI
- 4 SVI
- 5 Ensuring correctness of SVI
- 6 Conclusion

KL divergence

Status so far:

- given model P and parameterized guide Q_θ , we have defined the distributions p and q_θ that they induce over random data-bases
- we next need to establish a **measure of dissimilarity** between p and q_θ , to be able to define the optimization objective of SVI

Definition: KL divergence (Kullback-Leibler divergence)

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

$$\mathbf{D}_{\text{KL}}(p_0||p_1) = \mathbb{E}_{p_0} \left(\log \frac{p_0}{p_1} \right) = \int_{\mathcal{X}} p_0(x) \log \frac{p_0(x)}{p_1(x)} dx$$

In the discrete case:

$$\mathbf{D}_{\text{KL}}(p_0||p_1) = \sum_{x \in \mathcal{X}} p_0(x) \log \frac{p_0(x)}{p_1(x)}$$

Some properties of KL divergence

Main properties, for a given measurable space with measure $|\cdot|$:

- ① it is **positive**: for all p, q , $\mathbf{D}_{\text{KL}}(p||q) \geq 0$
- ② it is **null if and only if its arguments are equal almost everywhere**:

$$\mathbf{D}_{\text{KL}}(p||q) = 0 \iff |x| \neq 0 \implies p(x) = q(x)$$

- ③ it is **not symmetric**, which means that in general $\mathbf{D}_{\text{KL}}(p||q) \neq \mathbf{D}_{\text{KL}}(q||p)$
- ④ it **does not satisfy the triangular inequality**, which means that, in general, the inequality $\mathbf{D}_{\text{KL}}(p_0||p_2) \leq \mathbf{D}_{\text{KL}}(p_0||p_1) + \mathbf{D}_{\text{KL}}(p_1||p_2)$ does not hold

Due to the last two points it is called *divergence*, and not **distance**

Examples, over $X = \{0, 1\}$:

- p_0 defined by $p_0(0) = \frac{1}{2}$ and $p_0(1) = \frac{1}{2}$
- p_1 defined by $p_1(0) = \frac{1}{5}$ and $p_1(1) = \frac{4}{5}$
- p_2 defined by $p_2(0) = \frac{1}{10}$ and $p_2(1) = \frac{9}{10}$

Then:

- $\mathbf{D}_{\text{KL}}(p_0||p_1) \approx 0.22$ but $\mathbf{D}_{\text{KL}}(p_1||p_0) \approx 0.19$
- $\mathbf{D}_{\text{KL}}(p_0||p_2) \approx 0.51$ but $\mathbf{D}_{\text{KL}}(p_0||p_1) + \mathbf{D}_{\text{KL}}(p_1||p_2) \approx 0.27$

SVI objective

Setup:

- model P ; defines a probability distribution over sampled and observed random variables noted $p(z, x)$ where:
 - z : sampled random variables (ultimately represented in ρ)
 - x : observed random variables (ultimately not represented in ρ)
- guide Q , with a set of variables identified as parameters and noted t ; given a value assignment $t \mapsto \theta$, defines a probability distribution $q_\theta(z)$ over sampled random variables

We defined: $\mathcal{M}(p, S) = \int \rho(d\rho) (\mathbb{1}_{[\rho \in S]} \cdot \mathcal{D}[\![p]\!](\mu_I, \rho))$

It can be **normalized** into a **probability measure** iff $\mathcal{M}(p, \mathbb{P}) \in \mathbb{R}^{+*}$

Objective:

- beforehand, fix the family of programs q_θ as potential approximants of C
 - 1 with $\mathcal{M}(q_\theta, \mathbb{P}) = 1$, which is ensured **if q_θ always terminates**
 - 2 with density 1, which is ensured **if it contains no occurrence of observe**
- compute optimal θ to minimize

$$\mathbf{D}_{\text{KL}}(\mathcal{D}[\![q_\theta]\!](\mu_I, \rho), \mathcal{D}[\![C]\!](\mu_I, \rho))$$

SVI optimization objective

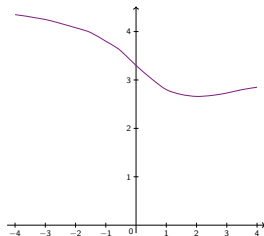
Inference goal

Compute an ideal value of θ that makes q_θ as close to p as possible, using an **optimization algorithm**

Application to the **inference problem**
two distributions over sampled variables z :

- $p(z|x)$:
posterior probability distribution over z defined by the model
(with the observation x)
- $q_\theta(z)$:
guide probability distribution
(parameterized by θ)

Plot of $\mathbf{D}_{\text{KL}}(q_\theta(z), p(z|x))$ as a function of θ :



Optimization objective: $\operatorname{argmin}_{\theta} \mathbf{D}_{\text{KL}}(q_\theta(z), p(z|x))$

Next step: achieve a **stochastic approximation of the gradient** of \mathbf{D}_{KL}

Definition of the optimization objective

We seek for θ so as to minimize:

$$\begin{aligned}
 & \mathbf{D}_{\text{KL}}(q_{\theta}(z), p(z|x)) \\
 &= \int_{\mathcal{Z}} q_{\theta}(z) \log \frac{q_{\theta}(z)}{p(z|x)} dz \\
 &= \int_{\mathcal{Z}} q_{\theta}(z) \log \frac{q_{\theta}(z)p(x)}{p(z,x)} dz && \text{by def. of conditional prob} \\
 &= \int_{\mathcal{Z}} q_{\theta}(z) \log p(x) dz + \int_{\mathcal{Z}} q_{\theta}(z) \log \frac{q_{\theta}(z)}{p(z,x)} dz \\
 &= \log p(x) \cdot \int_{\mathcal{Z}} q_{\theta}(z) dz + \int_{\mathcal{Z}} q_{\theta}(z) \log \frac{q_{\theta}(z)}{p(z,x)} dz \\
 &= \log p(x) \cdot 1 + \int_{\mathcal{Z}} q_{\theta}(z) \log \frac{q_{\theta}(z)}{p(z,x)} dz && \text{as } q_{\theta} \text{ is a probability} \\
 &= \log p(x) - \mathcal{L}(\theta)
 \end{aligned}$$

where $\mathcal{L}(\theta)$ is called the ELBO:

Evidence Lower Bound (ELBO)

The **ELBO** is defined as $\mathcal{L}(\theta) = \int_{\mathcal{Z}} q_{\theta}(z) \log \frac{p(z,x)}{q_{\theta}(z)} dz$.

Since $\mathbf{D}_{\text{KL}}(q_{\theta}(z), p(z|x)) + \mathcal{L}(\theta) = \log p(x)$ and $\log p(x)$ does not depend in θ ,

Minimizing $\mathbf{D}_{\text{KL}}(q_{\theta}(z), p(z|x))$ is equivalent to maximizing $\mathcal{L}(\theta)$

Gradient of the ELBO... (1)

To perform **gradient ascent** in order to **maximize ELBO**, we first need to evaluate this gradient, and formulate it **as an expectation**:

$$\begin{aligned}\nabla_{\theta} \mathcal{L}(\theta) &= \int_{\mathcal{Z}} \nabla_{\theta} \left[q_{\theta}(z) \log \frac{p(z, x)}{q_{\theta}(z)} dz \right] \quad \text{under assumption (later)} \\ &= \int_{\mathcal{Z}} (\nabla_{\theta} q_{\theta}(z)) \log \frac{p(z, x)}{q_{\theta}(z)} dz + \int_{\mathcal{Z}} q_{\theta}(z) \nabla_{\theta} \left(\log \frac{p(z, x)}{q_{\theta}(z)} \right) dz\end{aligned}$$

Let us study the second term more:

$$\begin{aligned}& \int_{\mathcal{Z}} q_{\theta}(z) \nabla_{\theta} \left(\log \frac{p(z, x)}{q_{\theta}(z)} \right) dz \\ &= \int_{\mathcal{Z}} q_{\theta}(z) \nabla_{\theta} (\log p(z, x) - \log q_{\theta}(z)) dz \\ &= \int_{\mathcal{Z}} q_{\theta}(z) \nabla_{\theta} (\log p(z, x)) dz - \int_{\mathcal{Z}} q_{\theta}(z) \nabla_{\theta} (\log q_{\theta}(z)) dz \\ &= - \int_{\mathcal{Z}} q_{\theta}(z) \nabla_{\theta} (\log q_{\theta}(z)) dz \quad (\text{the gradient of a constant is null}) \\ &= - \int_{\mathcal{Z}} q_{\theta}(z) \frac{\nabla_{\theta} q_{\theta}(z)}{q_{\theta}(z)} dz \\ &= - \int_{\mathcal{Z}} \nabla_{\theta} q_{\theta}(z) dz \\ &= - \nabla_{\theta} \left[\int_{\mathcal{Z}} q_{\theta}(z) dz \right] \\ &= - \nabla_{\theta} 1 \quad (\text{since } q_{\theta} \text{ defines a probability distribution}) \\ &= 0\end{aligned}$$

Gradient of the ELBO... (2)

We have shown:

$$\nabla_{\theta} \mathcal{L}(\theta) = \int_{\mathcal{Z}} (\nabla_{\theta} q_{\theta}(z)) \log \frac{p(z, x)}{q_{\theta}(z)} dz$$

We remark that $\nabla_{\theta} \log q_{\theta}(z) = \frac{\nabla_{\theta} q_{\theta}(z)}{q_{\theta}(z)}$

Thus, we can substitute $\nabla_{\theta} q_{\theta}(z) = q_{\theta}(z) \cdot \nabla_{\theta} \log q_{\theta}(z)$:

$$\begin{aligned} \nabla_{\theta} \mathcal{L}(\theta) &= \int_{\mathcal{Z}} (\nabla_{\theta} q_{\theta}(z)) \log \frac{p(z, x)}{q_{\theta}(z)} dz \\ &= \int_{\mathcal{Z}} q_{\theta}(z) \cdot \nabla_{\theta} \log q_{\theta}(z) \log \frac{p(z, x)}{q_{\theta}(z)} dz \\ &= \mathbb{E}_{q_{\theta}(z)} \left(\nabla_{\theta} \log q_{\theta}(z) \log \frac{p(z, x)}{q_{\theta}(z)} \right) \end{aligned}$$

This form, **as an expectation** is adapted to **stochastic approximation**
i.e., estimation based on a number of samples...

Stochastic approximation of the gradient of the ELBO

Based on $\nabla_{\theta} \mathcal{L}(\theta) = \mathbb{E}_{q_{\theta}(z)} \left(\nabla_{\theta} \log q_{\theta}(z) \log \frac{p(z, x)}{q_{\theta}(z)} \right)$ we may generate a sample ρ_0 from q_{θ} and produce

$$\mathbf{GrEst}_{\theta}(\rho_0) = (\nabla_{\theta} \log \mathcal{D}[\![q_{\theta}]\!](\mu_I, \rho_0)) \cdot \log \frac{\mathcal{D}[\![q_{\theta}]\!](\mu_I, \rho)}{\mathcal{D}[\![p]\!](\mu_I, \rho_0)}$$

Stochastic estimation repeats with **many samples**:

Stochastic approximant of gradient expectation formula

Using N samples from $\rho_0, \dots, \rho_{N-1}$ from distribution q_{θ} :

$$\mathbf{GrEst}_{\theta}(\rho) = \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta}(\rho_i) = \frac{1}{N} \sum_{i=0}^{N-1} (\nabla_{\theta} \log \mathcal{D}[\![q_{\theta}]\!](\mu_I, \rho_i)) \cdot \log \frac{\mathcal{D}[\![q_{\theta}]\!](\mu_I, \rho_i)}{\mathcal{D}[\![p]\!](\mu_I, \rho_i)}$$

Implementation:

- sampling executions from q_{θ} , e.g., by **rejection sampling**
- computation of $\mathcal{D}[\![q_{\theta}]\!]$ and $\mathcal{D}[\![p]\!]$ using **density semantics** based **instrumented implementation**; similarly, also accumulate $\nabla_{\theta}(\log \text{pdf})$ (table for classical distributions)
- usually sum log pdf rather than multiplying pdf

SVI algorithm based on gradient optimization

Fixed parameters:

- N number of samples per iterate for stochastic estimation
- λ : **learning rate**, typically small, e.g., $\lambda = 0.01$
- θ_{init} : **initial** value of the guide parameter (typically a rough guess)

Algorithm:

$$\left\{ \begin{array}{l} \text{select } \theta_0 := \theta_{\text{init}} \\ \text{repeat } K \text{ times} \\ \quad \text{sample } r_0, \dots, r_{N-1} \\ \quad \theta_{k+1} \leftarrow \theta_k - \lambda \cdot \frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta_k}(r_i) \\ \text{produce } \theta_K \end{array} \right.$$

Properties (under assumptions discussed shortly)

- $\frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta_k}(\rho_i)$ provides an **unbiased** estimate of the gradient at θ_k
- the algorithm **converges to a local maximum** θ of \mathcal{L} , i.e., a **local minimum** of $\mathbf{D}_{\text{KL}} \dots$

Outline

- 1 Introduction
- 2 Basic Intuition Underlying SVI
- 3 Semantics for SVI
- 4 SVI
- 5 Ensuring correctness of SVI
 - Behavior of SVI when assumptions do not hold
 - Discharging SVI assumptions using static analysis
- 6 Conclusion

Another model-guide pair

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:

- domain of σ in the **model**: $[0, 10]$
- domain of σ in the **guide**: \mathbb{R}
- thus, **KL-divergence is undefined**

(Example taken from the [Pyro webpage examples](#)...)

Issues possibly leading to undefinedness of KL-divergence

Absolute continuity requirement:

- definition of KL-divergence:

$$\mathbf{D}_{\text{KL}}(q_{\theta}, p) = \mathbb{E}_{q_{\theta}} \left(\log \frac{q_{\theta}}{p} \right) = \int_{\mathcal{X}} q_{\theta}(\mathrm{d}x) \log \frac{q_{\theta}(\mathrm{d}x)}{p(\mathrm{d}x)}$$

- absolute continuity requirement: model distribution p and guide distribution q should have **the same zero probability regions**
otherwise: **KL divergence is undefined**
- domain in **model** $[0, 10]$, in **guide** \mathbb{R}
leads to **the violation of absolute continuity assumption**
e.g., and **KL divergence is undefined**

Another possible issue: integrability

- $q_{\theta}(\mathrm{d}x) \log \frac{q_{\theta}(\mathrm{d}x)}{p(\mathrm{d}x)}$ may not be integrable
... even when 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 $\nabla \mathbf{D}_{\text{KL}}(q_{\theta}, p)$ 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

Unbiasedness conditions

Sufficient conditions (unbiasedness may hold in some cases where assumptions are violated, especially if locally so):

Theorem: unbiasedness of gradient estimate of KL divergence

If:

- ➊ absolute continuity: $\mathcal{D}[q_\theta](\mu_I)(\rho) \neq 0 \implies \mathcal{D}[p](\mu_I)(\rho) \neq 0$
- ➋ differentiability: $\theta \mapsto \mathcal{D}[q_\theta](\mu_I)(\rho)$ differentiable wrt all components
- ➌ boundness of KL divergence
- ➍ differentiability of KL divergence wrt all its arguments
- ➎ integral permutation conditions on KL divergence and guide density
 $\int \nabla \dots = \nabla \int \dots$

Then:

$$\mathbb{E}(\nabla_\theta \mathbf{D}_{\text{KL}}(\mathcal{D}[q_\theta](\mu_I), \mathcal{D}[C](\mu_I))) \equiv \frac{1}{N} \cdot \sum_{i=0}^{N-1} \mathbf{GrEst}_{\theta_k}(\rho_i)$$

Full version:

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

Outline

- 1 Introduction
- 2 Basic Intuition Underlying SVI
- 3 Semantics for SVI
- 4 SVI
- 5 Ensuring correctness of SVI
 - Behavior of SVI when assumptions do not hold
 - Discharging SVI assumptions using static analysis
- 6 Conclusion

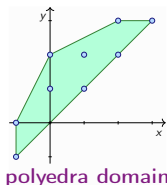
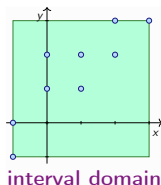
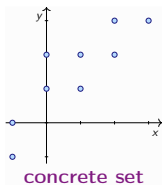
Abstract interpretation primer: state abstraction

Static analysis: interpretation using an **abstract domain**

Abstract domains (Cousot & Cousot, 1977)

- Families of **abstract predicates** adapted to static analysis
- **Compact** and **efficient** representations
- **Operations** for the static analysis of concrete operations
- Mapping from abstract to concrete: **concretization function** γ

Ex., numeric abstractions: abstraction of **sets of pairs of integers**



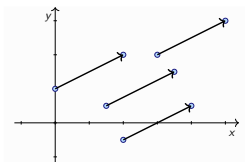
Abstract states **over-approximate** sets of concrete states

Abstract interpretation primer: analysis of post-conditions

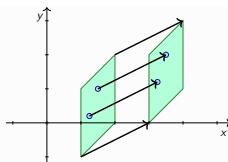
Computing sound abstract transformer

- **Conservative analysis** of concrete execution steps in the abstract
e.g., **assignments**, **condition tests**...
- May **lose precision**, will **never forget any behavior**
- Balance between **cost** and **precision**

Example: analysis of a **translation** with **octagons** (**exact!**)



concrete transformation



abstract transformation

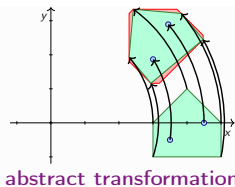
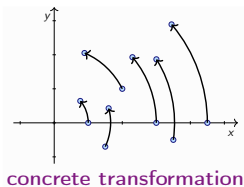
Soundness: all concrete behaviors are taken into account

Abstract interpretation primer: analysis of post-conditions

Computing sound abstract transformer

- **Conservative analysis** of concrete execution steps in the abstract
e.g., **assignments**, **condition tests**...
- May **lose precision**, will **never forget any behavior**
- Balance between **cost** and **precision**

Example: analysis of a **40 deg. rotation** with **octagons** (**approximate!**)



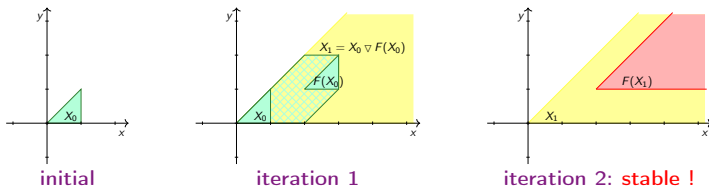
Soundness: all concrete behaviors are taken into account

Abstract interpretation primer: analysis of loops

Computing invariants about infinite executions with widening ∇

- **Widening** ∇ over-approximates \cup : **soundness guarantee**
- **Widening** ∇ guarantees the **termination of the analyses**
- Typical choice of ∇ : **remove unstable constraints**

Example: iteration of the translation $(2, 1)$, with **octagons**



Soundness: all concrete behaviors are taken into account

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 $\mathbb{D}^\#$ and:

- **concretization function** $\gamma : \mathbb{D}^\# \rightarrow \mathbb{D}$
where $\mathbb{D} = \mathbb{S} \rightarrow \mathcal{P}(\mathbb{S} \times \mathbb{R}^+ \times \mathbb{R}^+ \uplus \{\perp\})$
- **least element** $\perp^\#$ with $\gamma(\perp^\#) = \emptyset$
- **widening operator** $\nabla : \mathbb{D}^\# \times \mathbb{D}^\# \longrightarrow \mathbb{D}^\#$
over-approximating \cup
and enforcing termination on all sequences of abstract iterates
- **abstract composition** $\text{comp}^\# : \mathbb{D}^\# \times \mathbb{D}^\# \longrightarrow \mathbb{D}^\#$
soundness: $\forall g_0 \in \gamma(d_0^\#), \forall g_1 \in \gamma(d_1^\#), (g_0 \circ g_1) \in \gamma(\text{comp}^\#(d_0^\#, d_1^\#))$
- **abstract conditions, assignment, sample and score operations**
satisfying similar soundness conditions

Abstract interpreter

The definition of the abstract interpreter follows by induction over the syntax:

The abstract interpreter

$$\begin{aligned}
 \llbracket \text{skip} \rrbracket^\# &\triangleq \text{skip}^\# \\
 \llbracket \text{if } E \{ C_0 \} \text{ else } \{ C_1 \} \rrbracket^\# &\triangleq \text{cond}^\#(E)(\llbracket C_0 \rrbracket^\#, \llbracket C_1 \rrbracket^\#) \\
 \llbracket x := E \rrbracket^\# &\triangleq \text{assign}^\#(x, E) \\
 \llbracket x := \text{sample}_{\mathcal{N}}(k, E_1, E_2) \rrbracket^\# &\triangleq \text{sample}^\#(x, k, E_1, E_2) \\
 \llbracket C_0; C_1 \rrbracket^\# &\triangleq \text{comp}^\#(\llbracket C_1 \rrbracket^\#, \llbracket C_0 \rrbracket^\#) \\
 \llbracket \text{observe}_{\mathcal{N}}(E_0, E_1, E_2) \rrbracket^\# &\triangleq \text{score}^\#(E_0, E_1, E_2) \\
 \llbracket \text{while } E \{ C \} \rrbracket^\# &\triangleq \text{lfp}(\lambda d^\#. \text{cond}^\#(E)(\text{comp}^\#(d^\#, \llbracket C \rrbracket^\#), \text{skip}^\#))
 \end{aligned}$$

Exercise: provide the soundness condition for each of the operations

Soundness

The abstract semantics is sound in the sense that it **over-approximates** the effect of program concrete executions, in the sense of the density semantics:

Theorem: static analysis soundness

For all command C :

$$\llbracket C \rrbracket_{\mathcal{D}} \in \gamma(\llbracket C \rrbracket^{\#})$$

Proof: exercise!

The abstract semantics is not complete, and may not return the most precise abstraction for a given program...

First instance: static analysis for model/guide support match

Goal: **discharge model/guide support equality** (absolute continuity)

Abstraction

We define $\mathbb{D}^\#$ and γ by $\mathbb{D}^\# = \{\perp^\#, \top^\#\} \uplus \mathcal{P}(\mathbb{K})$ and:

$$\begin{aligned} \gamma : \quad \perp^\# &\longmapsto \lambda((\mu, \rho), w, p) \cdot \perp \\ \top^\# &\longmapsto \mathbb{D} \\ K(\subseteq \mathbb{K}) &\longmapsto \{g \in \mathbb{D} \mid \\ &\quad [\forall((\mu, \rho), w, p), \mu', w', p' \\ &\quad \quad g((\mu, \rho), w, p) = (\mu', \emptyset, w', p') \implies \text{Dom}(\rho) = K]\} \end{aligned}$$

A few transfer functions:

$$\begin{aligned} \text{comp}^\#(\perp^\#, d^\#) &= \text{comp}^\#(d^\#, \perp^\#) = \perp^\# \\ \text{comp}^\#(\top^\#, d^\#) &= \text{comp}^\#(d^\#, \top^\#) = \top^\# \\ \text{comp}^\#(K_0, K_1) &= \begin{cases} K_0 \uplus K_1 & \text{if } K_0 \cap K_1 = \emptyset \\ \top^\# & \text{otherwise} \end{cases} \\ \text{sample}^\#(x, k, E_0, E_1) &= \{k\} \\ \text{score}^\#(x, E_0, E_1, E_2) &= \emptyset \end{aligned}$$

First instance: static analysis for model/guide support match

Goal: **discharge model/guide support equality** (absolute continuity)

Abstraction

We define $\mathbb{D}^\#$ and γ by $\mathbb{D}^\# = \{\perp^\#, \top^\#\} \uplus \mathcal{P}(\mathbb{K})$ and:

$$\begin{aligned} \gamma : \quad \perp^\# &\longmapsto \lambda((\mu, \rho), w, p) \cdot \perp \\ \top^\# &\longmapsto \mathbb{D} \\ K(\subseteq \mathbb{K}) &\longmapsto \{g \in \mathbb{D} \mid \\ &\quad [\forall((\mu, \rho), w, p), \mu', w', p' \\ &\quad \quad g((\mu, \rho), w, p) = (\mu', \emptyset, w', p') \implies \text{Dom}(\rho) = K]]\} \end{aligned}$$

Example analysis:

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

Then: $\llbracket \text{model} \rrbracket^\# = \{v\}$

First instance: static analysis for model/guide support match

Goal: **discharge model/guide support equality** (absolute continuity)

Abstraction

We define $\mathbb{D}^\#$ and γ by $\mathbb{D}^\# = \{\perp^\#, \top^\#\} \uplus \mathcal{P}(\mathbb{K})$ and:

$$\begin{aligned} \gamma : \quad \perp^\# &\longmapsto \lambda((\mu, \rho), w, p) \cdot \perp \\ \top^\# &\longmapsto \mathbb{D} \\ K(\subseteq \mathbb{K}) &\longmapsto \{g \in \mathbb{D} \mid \\ &\quad [\forall((\mu, \rho), w, p), \mu', w', p' \\ &\quad \quad g((\mu, \rho), w, p) = (\mu', \emptyset, w', p') \implies \text{Dom}(\rho) = K]]\} \end{aligned}$$

Generalization: case where we consider **many distributions** (and not only normal distributions)

- the basic abstraction will not work:
sampling k from a normal distribution and sampling k from a Bernoulli distribution yield **distinct supports**
- new abstraction: $\mathbb{K} \rightarrow \text{Distributions} \uplus \{\perp^\#, \top^\#\}$

Second instance: static analysis for guide differentiability

Goal: **discharge differentiability properties**

Exercise:

- abstraction choice
- transfer functions

Outline

- 1 Introduction
- 2 Basic Intuition Underlying SVI
- 3 Semantics for SVI
- 4 SVI
- 5 Ensuring correctness of SVI
- 6 Conclusion**

Conclusion

Main ideas to remember from this lecture:

- SVI turns an inference problem into an **optimization problem**, to select an (possibly) optimal program in a parameterized family
- optimization only works when **a series of assumptions hold**

A few suggestions for **reading**:

- on **probabilistic semantics**:

Dexter Kozen,

Semantics of Probabilistic Programs.

Journal of Computing Systems Science(1981)

- on **SVI**, among other inference techniques:

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

An Introduction to Probabilistic Programming.

(2018)

- on **correctness of SVI**:

Wonyeol Lee, Hangyeol Yu, Xavier Rival, Hongseok Yang,

Towards verified stochastic variational inference for probabilistic programs,

PACM-POPL (2020)