Towards Verified Stochastic Variational Inference for Probabilistic Programs

Hongseok Yang
KAIST, South Korea

Joint with Wonyeol Lee and Hangyeol Yu (KAIST), and Xavier Rival (INRIA/ENS/CNRS)
Nontrivial assumptions are often made implicitly by ML algorithms, such as variational inference algo.

Be careful.
High-level message 2

Good research opportunity for PL/SE/Verification
— How to check those assumptions automatically?
Stochastic variational inference, and some pitfalls
def p(): // model_1
    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.)
```python
def p(): // model_1
    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.)
```
def p(): // model_1
    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.)
def p(): // model_1
    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.)
```python
def p():  # model_1
    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.)
```
def p(): // model_1
    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.)

def q_\theta(): // guide_1
    \theta = pyro.param("\theta", 0.)
    v = pyro.sample("v", Normal(\theta, 1.))
def p(): // model_1
    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.)

≈

def q_θ(): // guide_1
    θ = pyro.param("θ", 0.)
    v = pyro.sample("v", Normal(θ, 1.))
def p(): // model_1
    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.)

# define qθ:
θ = pyro.param("θ", 0.)
v = pyro.sample("v", Normal(θ, 1.))

qθ with θ found by optimisation
Typical optimisation objective:

$$\text{argmin}_\theta \text{KL}[q_\theta(z) \parallel p(z|x)]$$

where $\text{KL}[q_\theta(z) \parallel p(z|x)] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))]$. 
Typical optimisation objective:

$$\text{argmin}_{\theta} \text{KL}[q_{\theta}(z) \| p(z|x)]$$

where $\text{KL}[q_{\theta}(z) \| p(z|x)] = \mathbb{E}_{q_{\theta}(z)}[\log (q_{\theta}(z)/p(z|x))]$. 
Typical optimisation objective:

$$\text{argmin}_\theta \text{KL}[q_\theta(z) \| p(z|x)]$$

where  $$\text{KL}[q_\theta(z)\| p(z|x)] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))].$$
Typical optimisation objective:

$$\arg\min_{\theta} \text{KL}[q_{\theta}(z) \| p(z|x)]$$

where $$\text{KL}[q_{\theta}(z) \| p(z|x)] = \mathbb{E}_{q_{\theta}(z)}[\log (q_{\theta}(z)/p(z|x))]$$.

Optimisation by gradient descent:

$$\theta_{n+1} \leftarrow \theta_n - 0.01 \times \nabla_{\theta} \text{KL}[q_{\theta}(z) \| p(z|x)]_{\theta=\theta_n}$$
Typical optimisation objective:

\[
\arg\min_\theta \mathrm{KL}[q_\theta(z) \parallel p(z|x)]
\]

where \( \mathrm{KL}[q_\theta(z) \parallel p(z|x)] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \).

Optimisation by gradient descent:

\[
\theta_{n+1} \leftarrow \theta_n - 0.01 \times \nabla_\theta \mathrm{KL}[q_\theta(z) \parallel p(z|x)]_{\theta=\theta_n}
\]
Typical optimisation objective:

$$\text{argmin}_\theta \text{KL}[q_\theta(z) \parallel p(z|x)]$$

where $$\text{KL}[q_\theta(z) \parallel p(z|x)] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))]$$.

Issue 1: Undefined KL

Optimisation by gradient descent:

$$\theta_{n+1} \leftarrow \theta_n - 0.01 \times \nabla_\theta \text{KL}[q_\theta(z) \parallel p(z|x)]_{\theta=\theta_n}$$
Typical optimisation objective:

\[ \arg\min_{\theta} \text{KL}[q_{\theta}(z) \parallel p(z|x)] \]

where \( \text{KL}[q_{\theta}(z) \parallel p(z|x)] = \mathbb{E}_{q_{\theta}(z)}[\log \left( \frac{q_{\theta}(z)}{p(z|x)} \right)] \).

Optimisation by gradient descent:

\[ \theta_{n+1} \leftarrow \theta_n - 0.01 \times \nabla_{\theta} \text{KL}[q_{\theta}(z) \parallel p(z|x)]_{\theta=\theta_n} \]
Typical optimisation objective:

\[ \text{argmin}_\theta \text{KL}[q_\theta(z) \parallel p(z|x)] \]

where \( \text{KL}[q_\theta(z) \parallel p(z|x)] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \).

Optimisation by gradient descent:

\[ \theta_{n+1} \leftarrow \theta_n - 0.01 \times \nabla_{\theta} \text{KL}[q_\theta(z) \parallel p(z|x)]_{\theta=\theta_n} \]

**Issue 1: Undefined KL**
Typical optimisation objective:

$$\text{argmin}_\theta \text{KL}[q_\theta(z) \parallel p(z|x)]$$

where $\text{KL}[q_\theta(z) \parallel p(z|x)] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))]$.

Optimisation by gradient descent:

$$\theta_{n+1} \leftarrow \theta_n - 0.01 \times \nabla_\theta \text{KL}[q_\theta(z) \parallel p(z|x)]_{\theta=\theta_n}$$

Issue 1: Undefined KL

Issue 2: Non-differentiable KL
Typical optimisation objective:

$$\arg\min_\theta KL[q_\theta(z) \parallel p(z|x)]$$

where $$KL[q_\theta(z) \parallel p(z|x)] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))]$$.

Optimisation by gradient descent:

$$\theta_{n+1} \leftarrow \theta_n - 0.01 \times \nabla_\theta KL[q_\theta(z) \parallel p(z|x)]_{\theta=\theta_n}$$

Issue 1: Undefined KL

Issue 2: Non-differentiable KL

Issue 3: Wrong estimate
Issues

1. Undefined $\text{KL}[q_\theta(z) || p(z|x)]$.

2. Non-differentiable $\text{KL}[q_\theta(z) || p(z|x)]$.

3. Wrong estimate.
Issue 1: Undefined KL

\[ KL[q_\theta \parallel p] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \]

\[ = \int dz \ (q_\theta(z) \log (q_\theta(z)/p(z|x))) \]
Issue 1: Undefined KL

\[ KL[q_\theta \parallel p] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \]
\[ = \int dz \ (q_\theta(z) \log (q_\theta(z)/p(z|x))) \]

Two reasons for being undefined.
**Issue 1: Undefined KL**

\[
KL[q_\theta \parallel p] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \\
= \int dz \ (q_\theta(z) \log (q_\theta(z)/p(z|x)))
\]

Two reasons for being undefined.

- **Bad integrand** — \( p(z|x)=0 \) & \( q_\theta(z)\neq0 \) for some \( z \).
Issue 1: Undefined KL

\[ KL[q_\theta \parallel p] = \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \]

\[ = \int dz \ (q_\theta(z) \log (q_\theta(z)/p(z|x))) \]

Two reasons for being undefined:

• Bad integrand — \( p(z|x)=0 \) & \( q_\theta(z)\neq0 \) for some \( z \).

• Bad integral — Not integrable.
Bayesian regression from Pyro webpage.

```python
def p(...): // model_br
    ...
    sigma = pyro.sample("sigma", Uniform(0., 10.))
    ...
    pyro.sample("obs", Normal(..., sigma), obs=...)

def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(\theta, 0.05))
```

\[ \text{KL}[q_\theta(z) \| p(z|x)] \text{ undefined.} \]
Bayesian regression from Pyro webpage.

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

def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(\theta, 0.05))
```

$\text{KL}[q_{\theta}(z) \| p(z|x)]$ undefined. **Bad Integrand.**
Bayesian regression from Pyro webpage.

```python
def p(...): // model_br
    ...
    sigma = pyro.sample("sigma", Uniform(0., 10.))
    ...
    pyro.sample("obs", Normal(..., sigma), obs=...)
```

```python
def q_θ(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(θ, 0.05))
```

$KL[q_θ(z)∥p(z|x)]$ undefined. Bad Integrand.

[Q] Fix it.
Bayesian regression from Pyro webpage.

```python
def p(...): // model_br
    ...
    sigma = pyro.sample("sigma", Uniform(0., 10.))
    ...
    pyro.sample("obs", Normal(..., sigma), obs=...)
```

```python
def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(0, 0.05))
```

\[ \text{KL}[q_\theta(z)\|p(z|x)] \] undefined. Bad Integrand.

[Q] Fix it.
Bayesian regression from Pyro webpage.

```python
def p(...): // model_br
    ...
    sigma = pyro.sample("sigma", Uniform(0., 10.))
    ...
    pyro.sample("obs", Normal(..., sigma), obs=...)

def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(..., 0.05))
```

\[ \text{KL}[q_\theta(z) \| p(z|x)] \text{ undefined.} \]

\[ \text{Bad Integrand.} \]

\[ \text{Not integrable.} \]

[Q] Fix it.
Bayesian regression from Pyro

\[ \int_0^{10} d\sigma \frac{c^2}{\sigma^2} = \infty \]

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

```
def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(0., 0.05))
```

\[ \text{KL}[q_\theta(z)\|p(z|x)] \text{ undefined.} \]
\[ \text{Bad Integrand.} \]
\[ \boxed{\text{Not integrable.}} \]

\[ \boxed{\text{Fix it.}} \]
Bayesian regression from Pyro webpage.

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

def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(\theta, 0.05))
```

\[\text{KL}[q_\theta(z) \| p(z|x)]\text{ undefined. Bad Integrand.} \]

[Q] Fix it.
Bayesian regression from Pyro webpage.

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

def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(\theta, 0.05))
```

\[ \text{KL}[q_\theta(z) \| p(z|x)] \text{ undefined.} \]

\[ \text{Bad Integrand.} \]

\[ \text{[Q]} \text{ Fix it.} \]
Bayesian regression from Pyro webpage.

$$\int_{-1}^{1} d\sigma \left( \frac{c^2}{\sigma^2} \right) = \infty$$

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

```
def q_\theta(...): // guide_br
    ...
    sigma = pyro.sample("sigma", Normal(\theta, 0.05))
```

KL[q_\theta(z)||p(z|x)] undefined.  

Q] Fix it.
Issue 2: Non-differentiable KL

$KL[q_\theta(z)\|p(z|x)]$ may fail to be differentiable wrt. $\theta$.
def p(): // **model_1**
    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.)
```python
def p(): // model_1
    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.)

def q_\theta(): // guide_1'
    \theta = pyro.param("\theta", 4.)
    v = pyro.sample("v", Uniform(\theta -1., \theta +1.))
```

![Graph showing the model (prior) and model (posterior) for the given distributions.](attachment:image.png)
def p():  // model_1
    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.)

def q_\theta():  // guide_1'
    \theta = pyro.param("\theta", 4.)
    v = pyro.sample("v", Uniform(\theta - 1., \theta + 1.))
def p(): // model_1
    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.)

def q_θ(): // guide_1'
    θ = pyro.param("θ", 4.)
    v = pyro.sample("v", Uniform(θ - 1., θ + 1.))
Issue 3: Wrong estimate

Supposed to be unbiased, but not.

That is,

$$\nabla_\theta \text{KL}[q_\theta(z) \| p(z|x)] \neq \mathbb{E}[\nabla_\theta \text{KL}[q_\theta(z) \| p(z|x)]]$$.

when we expect equality.
Score estimator

\[ \nabla_\theta \text{KL}[q_\theta(z)||p(z|x)] \n\]

\[ = (\nabla_\theta \log q_\theta(z_0)) \times \log(q_\theta(z_0)/p(z_0,x)) \]

where \( z_0 \) is sampled from \( q_\theta \).
Score estimator

\[ \nabla_\theta \text{KL}[q_\theta(z) || p(z|x)] \]

\[ = (\nabla_\theta \log q_\theta(z_0)) \times \log(q_\theta(z_0)/p(z_0,x)) \]

where \( z_0 \) is sampled from \( q_\theta \).

Thm: \( \nabla_\theta \text{KL}[q_\theta(z) || p(z|x)] = \mathbb{E} [\nabla_\theta \text{KL}[q_\theta(z) || p(z|x)]] \)
Score estimator

\[ \nabla_\theta \text{KL}[q_\theta(z) || p(z|x)] \]

\[ = \left( \nabla_\theta \log q_\theta(z_0) \right) \times \log(q_\theta(z_0)/p(z_0, x)) \]

where \( z_0 \) is sampled from \( q_\theta \).

Thm: \[ \nabla_\theta \text{KL}[q_\theta(z) || p(z|x)] = \mathbb{E} \left[ \nabla_\theta \text{KL}[q_\theta(z) || p(z|x)] \right] \]

if some requirements are met.
Proof of the theorem

\[ \nabla_{\theta} KL[q_\theta(z) || p(z|x)] \]

\[ = \nabla_{\theta} \int dz \ (q_\theta(z) \times \log (q_\theta(z)/p(z|x))) \]

\[ = \int dz \ (\nabla_{\theta}(q_\theta(z) \times \log (q_\theta(z)/p(z|x)))) \]

\[ \ldots \]

\[ = \mathbb{E}[(\nabla_{\theta} \log q_\theta(z_0)) \times \log(q_\theta(z_0)/p(z_0,x))] \]
Proof of the theorem

Interchange of integration and differentiation. Might fail.

\[ \nabla_\theta \text{KL}[q_\theta(z) \| p(z|x)] \]

\[ = \nabla_\theta \int dz \ (q_\theta(z) \times \log (q_\theta(z)/p(z|x))) \]

\[ = \int dz \ (\nabla_\theta (q_\theta(z) \times \log (q_\theta(z)/p(z|x)))) \]

...  

\[ = \mathbb{E}[(\nabla_\theta \log q_\theta(z_0)) \times \log(q_\theta(z_0)/p(z_0,x)))] \]
def p(): // **model_1**
    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.)

def q_θ(): // **guide_1’**
    θ = pyro.param("θ", 4.)
    v = pyro.sample("v", Uniform(θ - 1., θ + 1.))

---

**Graph:**

- **KL**
- **θ**
- **KL + const**

def p(): // model_1
    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.)

def q_Θ(): // guide_1'
    θ = pyro.param("θ", 4.)
    v = pyro.sample("v", Uniform(θ - 1., θ + 1.))

---

non-zero slope at θ=4
we may drop "sub", and call $X$

consider the smallest measurable function on the same space $\mathbb{R}$

The second relies on two constructions, product and disjoint union. Suppose that we are given

The Lebesgue integral to its volume in the usual sense.

```
def p(): // model_1
    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.)
```

```
def q_\theta(): // guide_1'
    \theta = pyro.param("\theta", 4.)
    v = pyro.sample("v", Uniform(\theta -1., \theta +1.))
```
def p(): // model_1
    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.)

def qθ(): // guide_1'
    θ = pyro.param("θ", 4.)
    v = pyro.sample("v", Uniform(θ -1., θ +1.))
def p(): // model_1
  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.)

def q_θ(): // guide_1'
  θ = pyro.param("θ", 4.)
  v = pyro.sample("v", Uniform(θ - 1., θ + 1.))

\[ \nabla_θ KL[q_θ(z) \| p(z|x)]_\theta=4 = (\nabla_θ \log q_θ(z_0))_\theta=4 \ldots \\
= (\nabla_θ \log 0.5) \ldots \]
Defining the model and guide functions:

```python
def p(): # model_1
    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.)

def q_θ(): # guide_1'
    θ = pyro.param("θ", 4.)
    v = pyro.sample("v", Uniform(θ - 1., θ + 1.))
```

The KL divergence and its computation:

\[ \nabla_θ KL[q_θ(z) || p(z|x)] \theta=4 = (\nabla_θ \log q_θ(z_0)) \theta=4 \ldots = (\nabla_θ \log 0.5) \ldots = 0 \]

In the context of the Lebesgue integral, for a function \( \varepsilon \) on a measurable space \( X \):

- \( \mu \) is the reference measure
- \( \varepsilon \) is the subprobability measure
- \( \lambda \) is the invariant

To its volume in the usual sense:

\[ \lambda = \mu \text{.} \]

Towards Verified Stochastic Variational Inference for Probabilistic Programs.
How to check that these bad cases don’t happen?
1. Undefined $KL[q_\theta(z) || p(z|x)]$.
   - $p(z|x)=0$ & $q_\theta(z) \neq 0$ for some $z$.
   - Not integrable.

2. Non-differentiable $KL[q_\theta(z) || p(z|x)]$.

3. Wrong gradient estimate.
   - Biased score estimator due to $\nabla_\theta \int ... \neq \int \nabla_\theta ...$
1. Undefined KL[q_θ(z)||p(z|x)].
   • p(z|x)=0 & q_θ(z)≠0 for some z.
   • Not integrable.

2. Non-differentiable KL[q_θ(z)||p(z|x)].

3. Wrong gradient estimate.
   • Biased score estimator due to ∇_θ ∫ ... ≠ ∫ ∇_θ ...
1. Undefined $\text{KL}[q_\theta(z)\| p(z|x)]$.

- $p(z|x) = 0 \& q_\theta(z) \neq 0$ for some $z$.

```python
def p(): // model_1
    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.)

def q_\theta(): // guide_1
    \theta = pyro.param("\theta", 0.)
    v = pyro.sample("v", Normal(\theta, 1.))
```
1. Undefined KL[q_\theta(z)||p(z|x)].

- p(z|x)=0 & q_\theta(z) \neq 0 for some z.

```python
def p(): // model_1
    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.)

def q_\theta(): // guide_1
    \theta = pyro.param("\theta", 0.)
    v = pyro.sample("v", Normal(\theta, 1.))
```
1. Undefined $\text{KL}[q_\theta(z) \| p(z|x)]$.
   - $p(z|x) = 0 \& q_\theta(z) \neq 0$ for some $z$.
   - Not integrable.

2. Non-differentiable $\text{KL}[q_\theta(z) \| p(z|x)]$.

3. Wrong gradient estimate.
   - Biased score estimator due to $\nabla_\theta \int ... \neq \int \nabla_\theta ...$
Sufficient condition

Assume $q_\theta(z), p(z,x)$ use only normal distributions.

$\mu, \sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu', \sigma'$ - mean, standard deviation in $p(z,x)$. 
Sufficient condition

Assume $q_\theta(z)$, $p(z,x)$ use only normal distributions.

$\mu, \sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu', \sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$. 
Sufficient condition

Assume $q_\theta(z)$, $p(z,x)$ use only normal distributions.

$\mu, \sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu', \sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$. 
Sufficient condition

Assume $q_\theta(z)$, $p(z,x)$ use only normal distributions.

$\mu$, $\sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu'$, $\sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu$, $\sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g,h$. 

def p(): // model_1
    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.)

def q_\theta(): // guide_1
    \theta = pyro.param("\theta", 0.)
    v = pyro.sample("v", Normal(\theta, 1.))

\mu, \sigma - mean, standard deviation in q_\theta(z).

\mu', \sigma' - mean, standard deviation in p(z,x).

1. \mu, \sigma are continuously differentiable wrt. \theta.

2. |\mu'(z)| \leq \exp(f(|z|)) for affine f.

3. \exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|)) for affine g,h.
Sufficient condition

Assume $q(\theta)(z)$, $p(z, x)$ use only normal distributions.

$\mu$, $\sigma$ - mean, standard deviation in $q(\theta)(z)$.

$\mu'$, $\sigma'$ - mean, standard deviation in $p(z, x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g, h$. 

```python
def p():  // model_1
    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.)

def q_\theta():  // guide_1
    \theta = pyro.param("\theta", 0.)
    v = pyro.sample("v", Normal(\theta, 1.)))
```
Sufficient condition

Assume \( q_\theta(z), p(z,x) \) use only normal distributions.

\( \mu, \sigma \) - mean, standard deviation in \( q_\theta(z) \).

\( \mu', \sigma' \) - mean, standard deviation in \( p(z,x) \).

\[ \begin{align*}
\n1. & \quad \mu, \sigma \text{ are continuously differentiable wrt. } \theta. \\
2. & \quad |\mu'(z)| \leq \exp(f(|z|)) \text{ for affine } f. \\
3. & \quad \exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|)) \text{ for affine } g,h.
\end{align*} \]

```python
def p(): // model_1
    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.)

def q_\theta(): // guide_1
    \theta = pyro.param("\theta", 0.)
    v = pyro.sample("v", Normal(\theta, 1.))
```
Sufficient condition

Assume \( q_{\theta}(z) \), \( p(z,x) \) use only normal distributions.

\( \mu, \sigma \) - mean, standard deviation in \( q_{\theta}(z) \).

\( \mu', \sigma' \) - mean, standard deviation in \( p(z,x) \).

1. \( \mu, \sigma \) are continuously differentiable wrt. \( \theta \).

2. \( |\mu'(z)| \leq \exp(f(|z|)) \) for affine \( f \).

3. \( \exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|)) \) for affine \( g,h \).
Sufficient condition

Assume $q_{\theta}(z), p(z,x)$ use only normal distributions.

$\mu, \sigma$ - mean, standard deviation in $q_{\theta}(z)$.

$\mu', \sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g,h$. 
μ, σ - mean, standard deviation in qθ(z).

μ’, σ’ - mean, standard deviation in p(z,x).

1. μ, σ are continuously differentiable wrt. θ.

2. |μ’(z)| ≤ exp(f(|z|)) for affine f.

3. exp(g(|z|)) ≤ |σ’(z)| ≤ exp(h(|z|)) for affine g,h.

def p(): // **model_br’**
    sigma = pyro.sample("sigma", Normal(0., 5.))
    pyro.sample("obs", Normal(0., abs(sigma)), obs=2.)

def qθ(): // **guide_br’**
    θ = pyro.param("θ", 2.)
    sigma = pyro.sample("sigma", Normal(θ, 0.05))
μ, σ - mean, standard deviation in \( q_\theta(z) \).

\( \mu', \sigma' \) - mean, standard deviation in \( p(z, x) \).

\( \checkmark \) 1. \( \mu, \sigma \) are continuously differentiable wrt. \( \theta \).

2. \( |\mu'(z)| \leq \exp(f(|z|)) \) for affine \( f \).

3. \( \exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|)) \) for affine \( g, h \).
Assume $q(\theta)(z)$, $p(z,x)$ use only normal distributions.

$\mu$, $\sigma$ - mean, standard deviation in $q(\theta)(z)$.

$\mu'$, $\sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu$, $\sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g,h$. 

```python
def p(): // model_br'
    sigma = pyro.sample("sigma", Normal(0., 5.))
    pyro.sample("obs", Normal(0., abs(sigma)), obs=2.)

def q_\theta(): // guide_br'
    \theta = pyro.param("\theta", 2.)
    sigma = pyro.sample("sigma", Normal(\theta, 0.05))
```
Assume $q_\theta(z)$, $p(z,x)$ use only normal distributions.

$\mu$, $\sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu'$, $\sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g,h$. 

```python
def p(): // model_br'
    sigma = pyro.sample("sigma", Normal(0., 5.))
    pyro.sample("obs", Normal(0., abs(sigma)), obs=2.)

def q_\theta(): // guide_br'
    \theta = pyro.param("\theta", 2.)
    sigma = pyro.sample("sigma", Normal(\theta, 0.05))
```

Sufficient condition

Assume $q_{\theta}(z)$, $p(z,x)$ use only normal distributions.

$\mu$, $\sigma$ - mean, standard deviation in $q_{\theta}(z)$.

$\mu'$, $\sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu$, $\sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g,h$. 
Assume $q_\theta(z)$, $p(z,x)$ use only normal distributions.

$\mu, \sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu', \sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g,h$. 

1. KL not integrable.
2. KL not differentiable.
3. Biased due to $\nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots$
Assume $q_{\theta}(z)$, $p(z,x)$ use only normal distributions.

$\mu$, $\sigma$ - mean, standard deviation in $q_{\theta}(z)$.

$\mu'$, $\sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g, h$. 

1. KL not integrable.

2. KL not differentiable.

3. Biased due to $\nabla_{\theta} \int \ldots \neq \int \nabla_{\theta} \ldots$
1. KL not integrable.
2. KL not differentiable.
3. Biased due to $\nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots$

Assume $q_\theta(z)$, $p(z,x)$ use only normal distributions.

$\mu$, $\sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu'$, $\sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu$, $\sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g,h$. 
1. KL not integrable.
2. KL not differentiable.
3. Biased due to $\nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots$

Assume $q_\theta(z), p(z,x)$ use only normal distributions.

$\mu, \sigma$ - mean, standard deviation in $q_\theta(z)$.

$\mu', \sigma'$ - mean, standard deviation in $p(z,x)$.

1. $\mu, \sigma$ are continuously differentiable wrt. $\theta$.

2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine $f$.

3. $\exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|))$ for affine $g, h$. 
1. KL not integrable.
2. KL not differentiable.
3. Biased due to

\[
\int dz \left( \mathcal{N}(z; \ldots) \cdot \exp(f(|z|)) \right) < \infty
\]
for all affine f

Assume \( q_\theta(z), p(z,x) \) use only normal distributions.

\( \mu, \sigma \) - mean, standard deviation in \( q_\theta(z) \).

\( \mu', \sigma' \) - mean, standard deviation in \( p(z,x) \).

1. \( \mu, \sigma \) are continuously differentiable wrt. \( \theta \).

2. \( |\mu'(z)| \leq \exp(f(|z|)) \) for affine f.

3. \( \exp(g(|z|)) \leq |\sigma'(z)| \leq \exp(h(|z|)) \) for affine g,h.
Useful in practice?
Our automatic verifier

• Works for Pyro programs.

• Proves the following bad cases don’t happen:

$$p(z|x)=0 \ & \ q_\theta(z)\neq0 \ \text{for some } z.$$ 

• Handles features of Python/PyTorch/Pyro, such as tensor broadcasting, but not all of them.
Table 1. Key features of the model-guide pairs from Pyro examples. LoC denotes the lines of code of model and guide. The columns “Total #” show the number of objects/commands of each type used in model and guide, and the columns “Total dimension” show the total dimension of tensors in model and guide, either sampled from sample or used inside score, as well as the dimension of \( \theta \) in guide.

<table>
<thead>
<tr>
<th>Name</th>
<th>Corresponding probabilistic model</th>
<th>LoC</th>
<th>Total # for plate sample</th>
<th>Total dimension</th>
<th>Total dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>br</td>
<td>Bayesian regression</td>
<td>27</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>csis</td>
<td>Compiled sequential importance sampling</td>
<td>31</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>lda</td>
<td>Latent Dirichlet allocation (LDA)</td>
<td>76</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>vae</td>
<td>Variational autoencoder (VAE)</td>
<td>91</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>sgdef</td>
<td>Sparse gamma deep exponential family</td>
<td>94</td>
<td>0</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>dmm</td>
<td>Deep Markov model</td>
<td>246</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ssvae</td>
<td>Semi-supervised VAE</td>
<td>349</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>air</td>
<td>Attend-infer-repeat (AIR)</td>
<td>410</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Analysed 8 representative Pyro programs from Pyro webpage.
Table 2. Analysis results on two benchmark sets. The column "Time" shows the analysis time in seconds; in (a), it is averaged over those model-guide pairs (in each category) for which our analyser does not crash. In both cases, the found correctness issues are subtle and were not known before.

Verification of probabilistic programs relying on model-guide pairs. Among the Pyro test suite, the analysis successfully verifies 31 examples among 39. Interestingly, two of these 31 successful validations, highlighted in Table 2 (a), correspond to cases that were tagged as "invalid model-guide pairs" in the Pyro git repository. Upon inspection, these two examples turn out to be correct. On the other hand, 8 examples from the Pyro test suite could not be verified due to the crashes of the analyser. One of these failures is due to the need to reason more precisely about the content of a for loop (e.g., using some partitioning techniques), and seven are due to the use of plates with subsampling, as ranges for for loops. Therefore these failures could be resolved using existing static analysis techniques and a more precise handling of the semantics of Python constructions.

Moreover, 6 Pyro examples (among the 8 that we considered) were verified successfully, which means all correct Pyro examples were verified. Finally, we corrected the two examples that were rejected due to invalid model-guide pairs, and these two examples were also successfully verified.

Analysis efficiency. The analysis returned within a second on each program in the Pyro test suite, and on most of the Pyro examples. In fact, the slowest analysis was observed on air, which was analysed within 5 seconds. Most of the Pyro examples sample from and score with distributions of very high dimension arranged in complex tensors, using nested for and plate's. While they are not large, they present a high degree of logical complexity, that is representative of realistic probabilistic programs. The fact that such programs get analysed within seconds shows that the analysis and the underlying abstract domain to describe zones, sampled dimensions, and distributions can generalise predicates quickly so that precise loop invariants can be computed.

9 RELATED WORK
As far as we know, the idea of using stochastic variational inference for probabilistic programs first appeared in [Wingate and Weber 2013]. When further insights into how to create generic (sometimes also called black-box) SVI engines were found [Kucukelbir et al. 2015, 2017; Ranganath et al. 2014], the idea was tried for realistic probabilistic programming languages, such as Stan [Kucukelbir et al. 2016].
Towards Verified Stochastic Variational Inference for Probabilistic Programs

(a) Results for Pyro test suite. 39 model-guide pairs are grouped into 7 categories, based on which type of plate objects are used. #Same (or #Different) denotes the number of model-guide pairs for which the output of our analyser, valid or invalid, is the same as (or different from) the documented output. #Crash denotes the number of pairs for which our analyser crashes.

(b) Results for Pyro examples. The column “Valid?” shows the output of our analysis, valid or invalid.

<table>
<thead>
<tr>
<th>Name</th>
<th>Valid?</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>br</td>
<td>x</td>
<td>0.006</td>
</tr>
<tr>
<td>csis</td>
<td>o</td>
<td>0.007</td>
</tr>
<tr>
<td>lda</td>
<td>x</td>
<td>0.014</td>
</tr>
<tr>
<td>vae</td>
<td>o</td>
<td>0.005</td>
</tr>
<tr>
<td>sgdef</td>
<td>o</td>
<td>0.070</td>
</tr>
<tr>
<td>dmm</td>
<td>o</td>
<td>0.536</td>
</tr>
<tr>
<td>ssvae</td>
<td>o</td>
<td>0.013</td>
</tr>
<tr>
<td>air</td>
<td>o</td>
<td>4.093</td>
</tr>
</tbody>
</table>

In both cases, the found correctness issues are subtle and were not known before. Verification of probabilistic programs relying on model-guide pairs. Among the Pyro test suite, the analysis successfully verifies 31 examples among 39. Interestingly, two of these 31 successful validations, highlighted in Table 2 (a), correspond to cases that were tagged as “invalid model-guide pairs” in the Pyro git repository. Upon inspection, these two examples turn out to be correct.

On the other hand, 8 examples from the Pyro test suite could not be verified due to the crashes of the analyser. One of these failures is due to the need to reason more precisely about the content of a for loop (e.g., using some partitioning techniques), and seven are due to the use of plates with subsampling, as ranges for for loops. Therefore these failures could be resolved using existing static analysis techniques and a more precise handling of the semantics of Python constructions.

Moreover, 6 Pyro examples (among the 8 that we considered) were verified successfully, which means all correct Pyro examples were verified. Finally, we corrected the two examples that were rejected due to invalid model-guide pairs, and these two examples were also successfully verified.

Analysis efficiency. The analysis returned within a second on each program in the Pyro test suite, and on most of the Pyro examples. In fact, the slowest analysis was observed on air, which was analysed within 5 seconds. Most of the Pyro examples sample from and score with distributions of very high dimension arranged in complex tensors, using nested for and plate’s. While they are not large, they present a high degree of logical complexity, that is representative of realistic probabilistic programs. The fact that such programs get analysed within seconds shows that the analysis and the underlying abstract domain to describe zones, sampled dimensions, and distributions can generalise predicates quickly so that precise loop invariants can be computed.

9 RELATED WORK

As far as we know, the idea of using stochastic variational inference for probabilistic programs first appeared in Wingate and Weber 2013. When further insights into how to create generic (sometimes also called black-box) SVI engines were found Kucukelbir et al. 2015, 2017; Ranganath et al. 2014, the idea was tried for realistic probabilistic programming languages, such as Stan Kucukelbir et al.
Towards Verified Stochastic Variational Inference for Probabilistic Programs

### Results for Pyro test suite

<table>
<thead>
<tr>
<th>Category</th>
<th>#Same</th>
<th>#Di</th>
<th>Crash Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>9</td>
<td>0</td>
<td>0.001</td>
</tr>
<tr>
<td>Nested</td>
<td>4</td>
<td>0</td>
<td>0.004</td>
</tr>
<tr>
<td>Single with plate</td>
<td>6</td>
<td>0</td>
<td>0.026</td>
</tr>
<tr>
<td>Nested with plate</td>
<td>2</td>
<td>0</td>
<td>0.002</td>
</tr>
<tr>
<td>Non-nested with plate</td>
<td>2</td>
<td>0</td>
<td>0.002</td>
</tr>
<tr>
<td>Nested for-plate &amp; with-plate</td>
<td>0</td>
<td>3</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Total: 29 2 8 0.003

(a) Results for Pyro test suite. 39 model-guide pairs are grouped into 7 categories, based on which type of plate objects are used. 

#Same (or #Di) denotes the number of model-guide pairs for which the output of our analyser, valid or invalid, is the same (or different from) the documented output. #Crash denotes the number of pairs for which our analyser crashes.

### Results for Pyro examples

<table>
<thead>
<tr>
<th>Name</th>
<th>Valid?</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>br</td>
<td>x</td>
<td>0.006</td>
</tr>
<tr>
<td>csis</td>
<td>o</td>
<td>0.007</td>
</tr>
<tr>
<td>lda</td>
<td>x</td>
<td>0.014</td>
</tr>
<tr>
<td>vae</td>
<td>o</td>
<td>0.005</td>
</tr>
<tr>
<td>sgdef</td>
<td>o</td>
<td>0.070</td>
</tr>
<tr>
<td>dmm</td>
<td>o</td>
<td>0.536</td>
</tr>
<tr>
<td>ssvae</td>
<td>o</td>
<td>0.013</td>
</tr>
<tr>
<td>air</td>
<td>o</td>
<td>4.093</td>
</tr>
</tbody>
</table>

(b) Results for Pyro examples. The column “Valid?” shows the output of our analysis, valid or invalid.

Table 2. Analysis results on two benchmark sets. The column “Time” shows the analysis time in seconds; in (a), it is averaged over those model-guide pairs (in each category) for which our analyser does not crash.

In both cases, the found correctness issues are subtle and were not known before.

Verification of probabilistic programs relying on model-guide pairs. Among the Pyro test suite, the analysis successfully verifies 31 examples among 39. Interestingly, two of these 31 successful validations, highlighted in Table 2 (a), correspond to cases that were tagged as “invalid model-guide pairs” in the Pyro git repository. Upon inspection, these two examples turn out to be correct.

On the other hand, 8 examples from the Pyro test suite could not be verified due to the crashes of the analyser. One of these failures is due to the need to reason more precisely about the content of a for loop (e.g., using some partitioning techniques), and seven are due to the use of plates with subsampling, as ranges for for loops. Therefore these failures could be resolved using existing static analysis techniques and a more precise handling of the semantics of Python constructions.

Moreover, 6 Pyro examples (among the 8 that we considered) were verified successfully, which means all correct Pyro examples were verified. Finally, we corrected the two examples that were rejected due to invalid model-guide pairs, and these two examples were also successfully verified.

Analysis efficiency. The analysis returned within a second on each program in the Pyro test suite, and on most of the Pyro examples. In fact, the slowest analysis was observed on air, which was analysed within 5 seconds. Most of the Pyro examples sample from and score with distributions of very high dimension arranged in complex tensors, using nested for and plate’s. While they are not large, they present a high degree of logical complexity, that is representative of realistic probabilistic programs. The fact that such programs get analysed within seconds shows that the analysis and the underlying abstract domain to describe zones, sampled dimensions, and distributions can generalise predicates quickly so that precise loop invariants can be computed.

### Related Work

As far as we know, the idea of using stochastic variational inference for probabilistic programs first appeared in [Wingate and Weber 2013]. When further insights into how to create generic (sometimes also called black-box) SVI engines were found [Kucukelbir et al. 2015, 2017; Ranganath et al. 2014], the idea was tried for realistic probabilistic programming languages, such as Stan [Kucukelbir et al. 2016].
Reference

The details can be found in our archive paper: