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)

High-level message 1

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

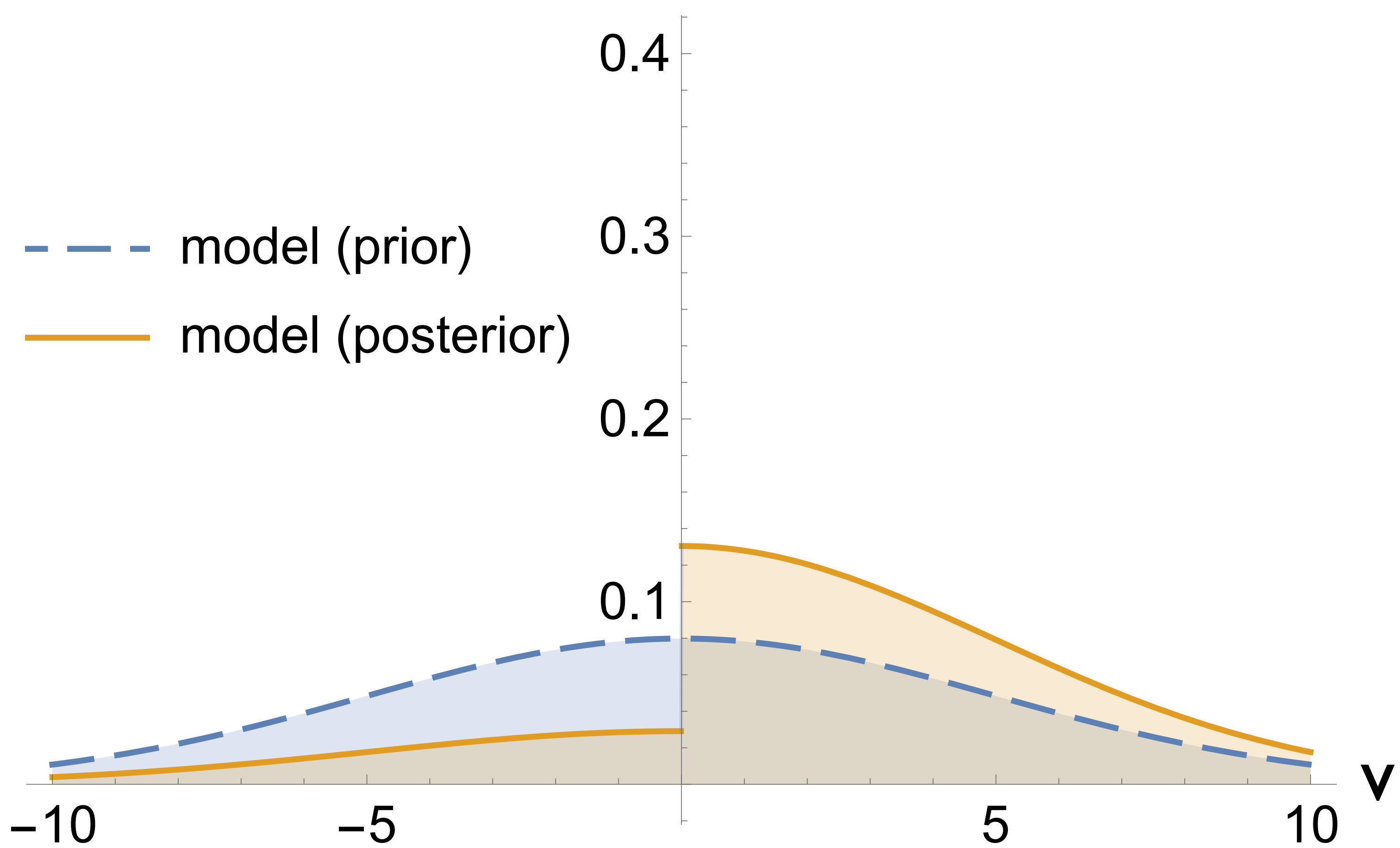
```
def p(): // model_1
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):  pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:  pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

```
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.)
```

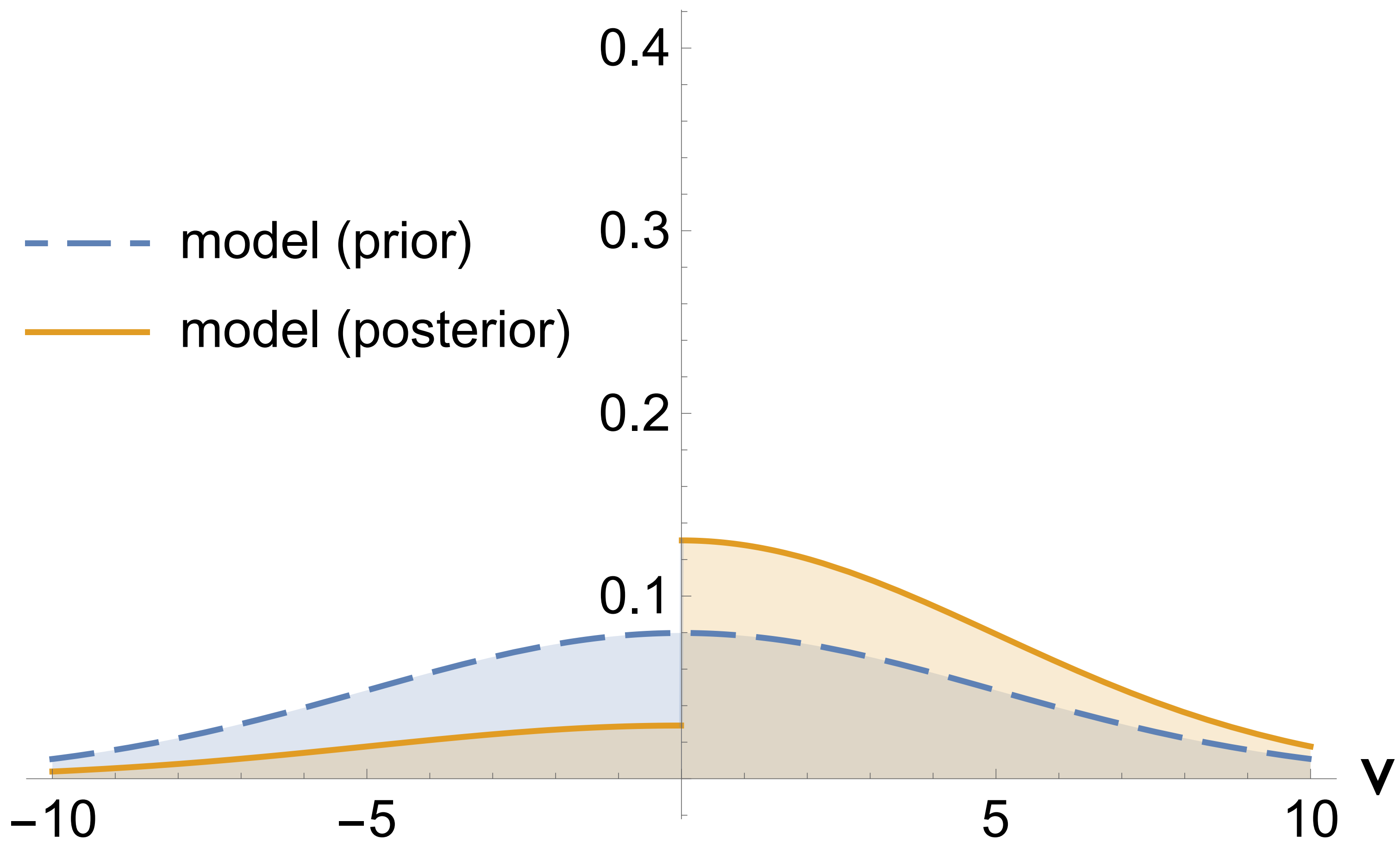
```
def p(): // model_1
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):  pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:  pyro.sample("obs", Normal(-2., 1.), obs=0.)
```



```
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.)
```

≈

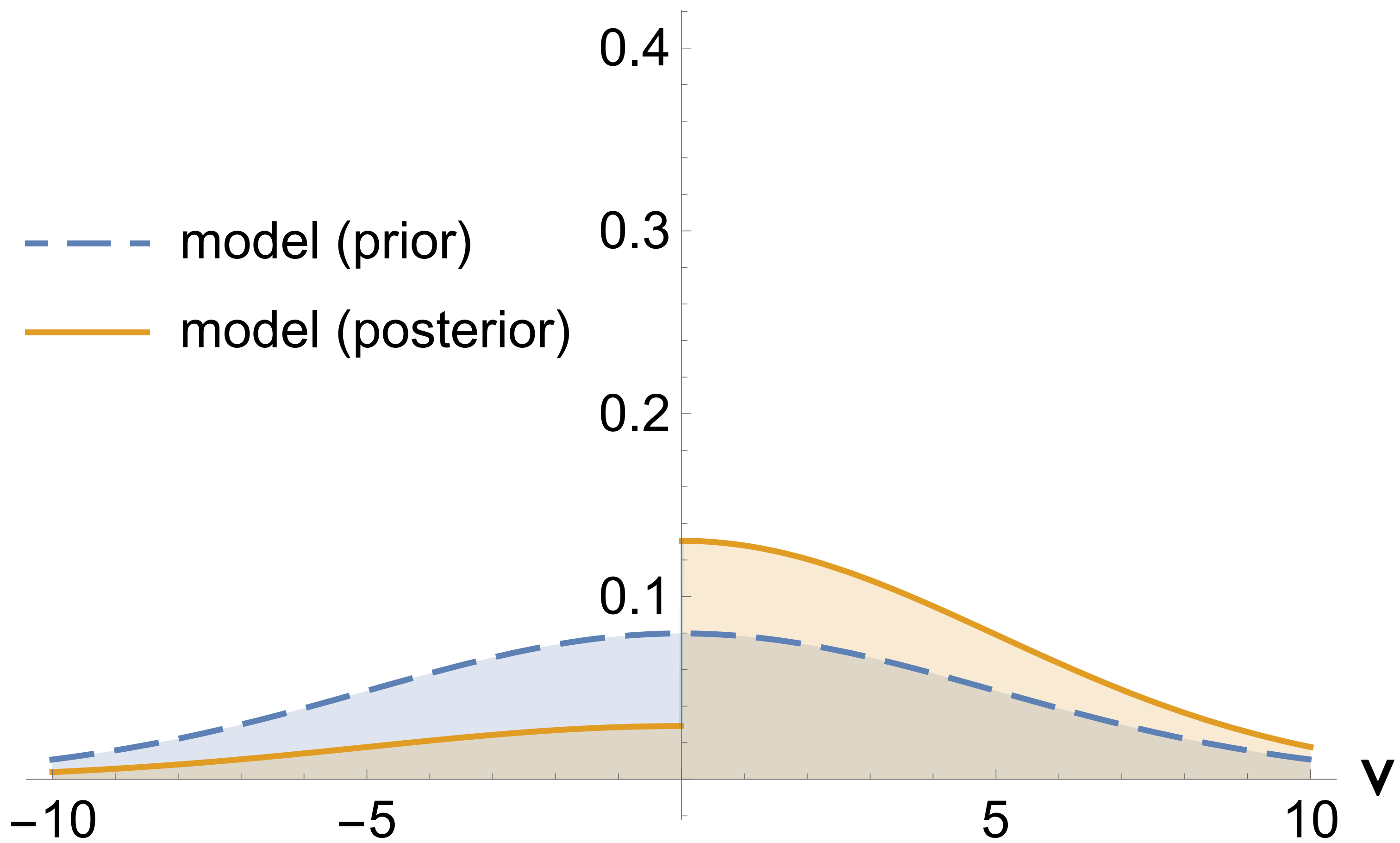
```
def qθ(): // guide_1
  θ = pyro.param("θ", 0.)
  v = pyro.sample("v", Normal(θ, 1.))
```



```
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.)
```

$\approx$

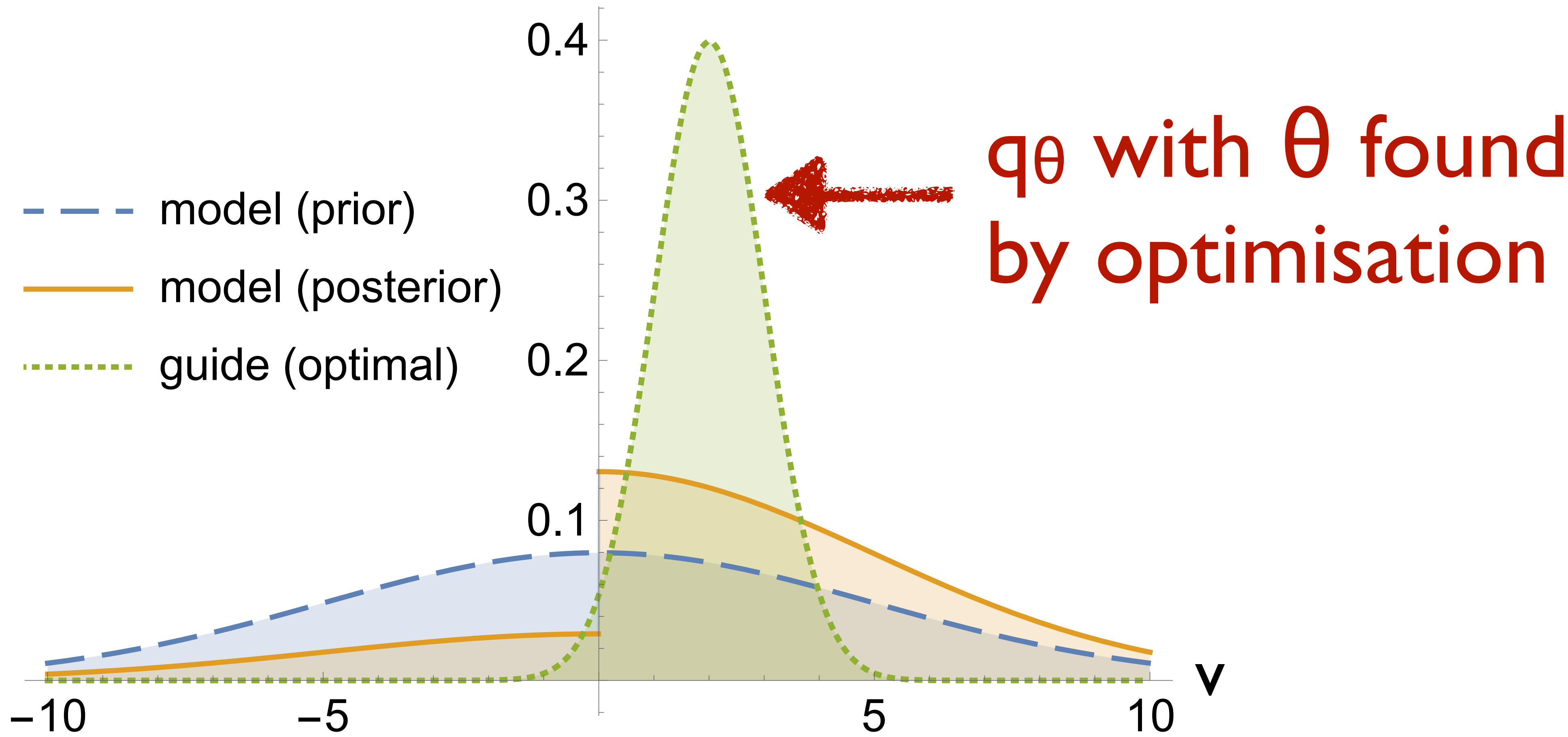
```
def qθ(): // guide_1
  θ = pyro.param("θ", 0.)
  v = pyro.sample("v", Normal(θ, 1.))
```



```
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.)
```

≈

```
def qθ(): // guide_1
   θ = pyro.param("θ", 0.)
   v = pyro.sample("v", Normal(θ, 1.))
```



Typical optimisation objective:

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

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

Typical optimisation objective:

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

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

Typical optimisation objective:

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

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

Typical optimisation objective:

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

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

Optimisation by gradient descent:

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

Typical optimisation objective:

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

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

Optimisation by gradient descent:

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

$\boxed{\text{Issue 1: Undefined KL}}$

Typical optimisation objective:

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

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

Optimisation by gradient descent:

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

Issue 1: Undefined KL

Typical optimisation objective:

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

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

Optimisation by gradient descent:

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

Issue 1: Undefined KL

Typical optimisation objective:

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

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

Optimisation by gradient descent:

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

$\boxed{\text{Issue 1: Undefined KL}}$

Typical optimisation objective:

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

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

Optimisation by gradient descent:

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

$\boxed{\text{Issue 2: Non-differentiable KL}}$

Issue 1: Undefined KL

Issue 3:
Wrong estimate

Typical optimisation objective:

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

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

Optimisation by gradient descent:

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

Issue 2: Non-differentiable KL

Issues

1. Undefined $KL[q_\theta(z)||p(z|x)]$.
2. Non-differentiable $KL[q_\theta(z)||p(z|x)]$.
3. Wrong estimate.

Issue 1: Undefined KL

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

$$= \int dz\,(q_\theta(z)\,\log\,(q_\theta(z)/p(z|x)))$$

Issue 1: Undefined KL

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

$$= \int dz\,(q_\theta(z)\,\log\,(q_\theta(z)/p(z|x)))$$

Two reasons for being undefined.

Issue 1: Undefined KL

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

$$= \int dz \, (q_\theta(z) \log (q_\theta(z)/p(z|x)))$$

Two reasons for being undefined.

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

Issue 1: Undefined KL

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

$$= \int dz\,(q_\theta(z)\,\log\,(q_\theta(z)/p(z|x)))$$

Two reasons for being undefined:

- Bad integrand — $p(z|x)=0$ & $q_\theta(z)\neq 0$ for some z.
- Bad integral — Not integrable.

Bayesian regression from Pyro webpage.

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

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

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

Bayesian regression from Pyro webpage.

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

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

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

Bayesian regression from Pyro webpage.

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

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

$KL[q_\theta(z)||p(z|x)]$ 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 qθ(...): // guide_br
  ...                                 Uniform(0., 10.)
  sigma = pyro.sample("sigma", Normal(0, 0.05))
```

KL[$q_\theta(z)||p(z|x)$] 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 qθ(...): // guide_br
  ...                                  Uniform(0., 10.)
  sigma = pyro.sample("sigma", Normal(0, 0.05))
```

Not integrable.

$KL[q_\theta(z)||p(z|x)]$ undefined. ~~Bad Integrand.~~

[Q] Fix it.

Bayesian regression from Pyro

$$\boxed{\int_0^{10} \mathrm{d}\sigma \, \frac{c^2}{\sigma^2} = \infty}$$

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

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

Uniform(0., 10.)

KL[$q_\theta(z)||p(z|x)$] undefined. ~~Bad Integrand.~~ Not integrable.

[Q] Fix it.

Bayesian regression from Pyro webpage.

```
def p(...): // model_br
  ...
                                        Normal(0., 5.0)
  sigma = pyro.sample("sigma", ~~Uniform(0., 10.)~~)
  ...
  pyro.sample("obs", Normal(..., ~~sigma~~), obs=...)
                                   abs(sigma)
```

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

Not integrable.

$KL[q_\theta(z)||p(z|x)]$ undefined. ~~Bad Integrand.~~

[Q] Fix it.

Bayesian regression from Pyro webpage.

def p(...): // model_br

  ...

  Normal(0., 5.0)

  sigma = pyro.sample("sigma", ~~Uniform(0., 10.)~~)

  ...

  pyro.sample("obs", Normal(..., ~~sigma~~), obs=...)

  abs(sigma)

def q_θ(...): // guide_br

  ...

  sigma = pyro.sample("sigma", Normal(θ, 0.05))

Not integrable.

KL[$q_\theta(z)||p(z|x)$] undefined. ~~Bad Integrand.~~

[Q] Fix it.

Bayesian regression fr

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

def p(...): // model_br

  ...

  Normal(0., 5.0)

  sigma = pyro.sample("sigma", ~~Uniform(0., 10.)~~)

  ...

  pyro.sample("obs", Normal(..., ~~sigma~~), obs=...)

  abs(sigma)

def q_θ(...): // guide_br

  ...

  sigma = pyro.sample("sigma", Normal(θ, 0.05))

Not integrable.

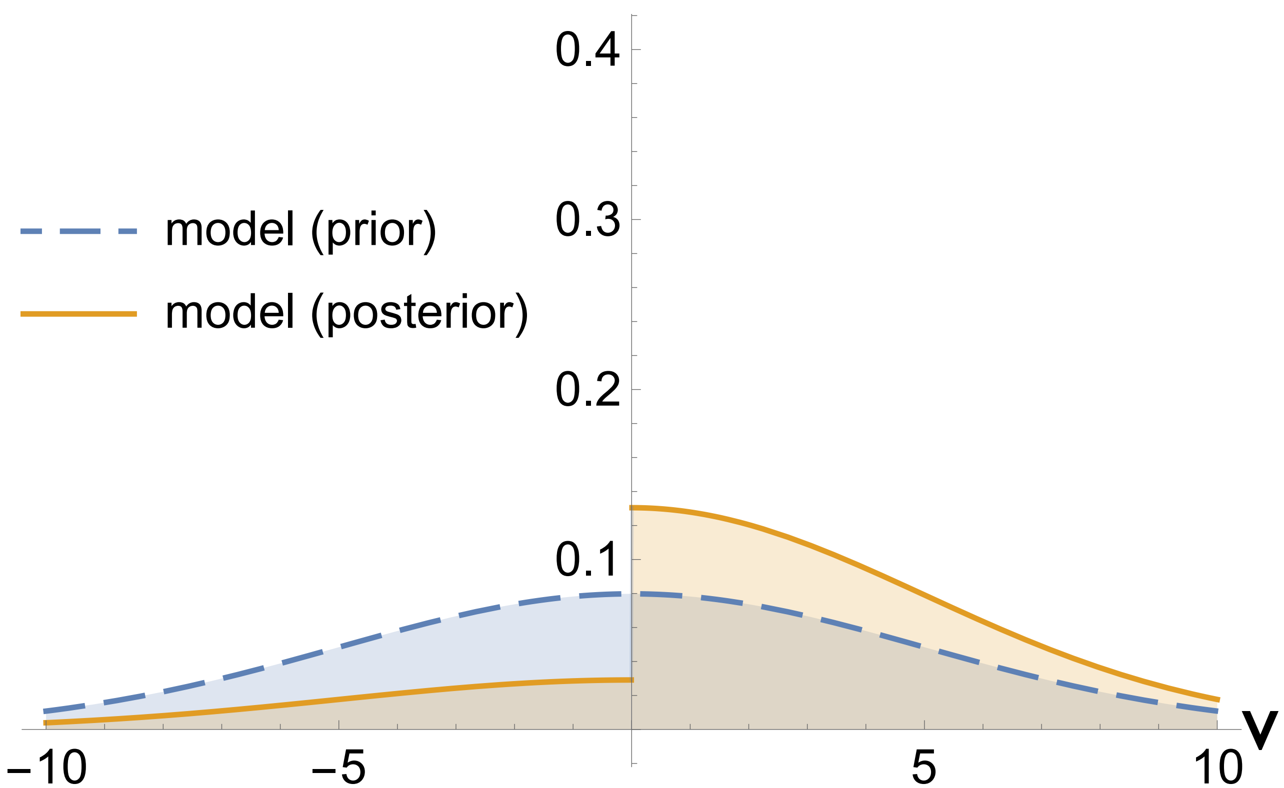
KL[$q_\theta(z)||p(z|x)$] undefined. ~~Bad Integrand.~~

[Q] Fix it.

Issue 2: Non-differentiable KL

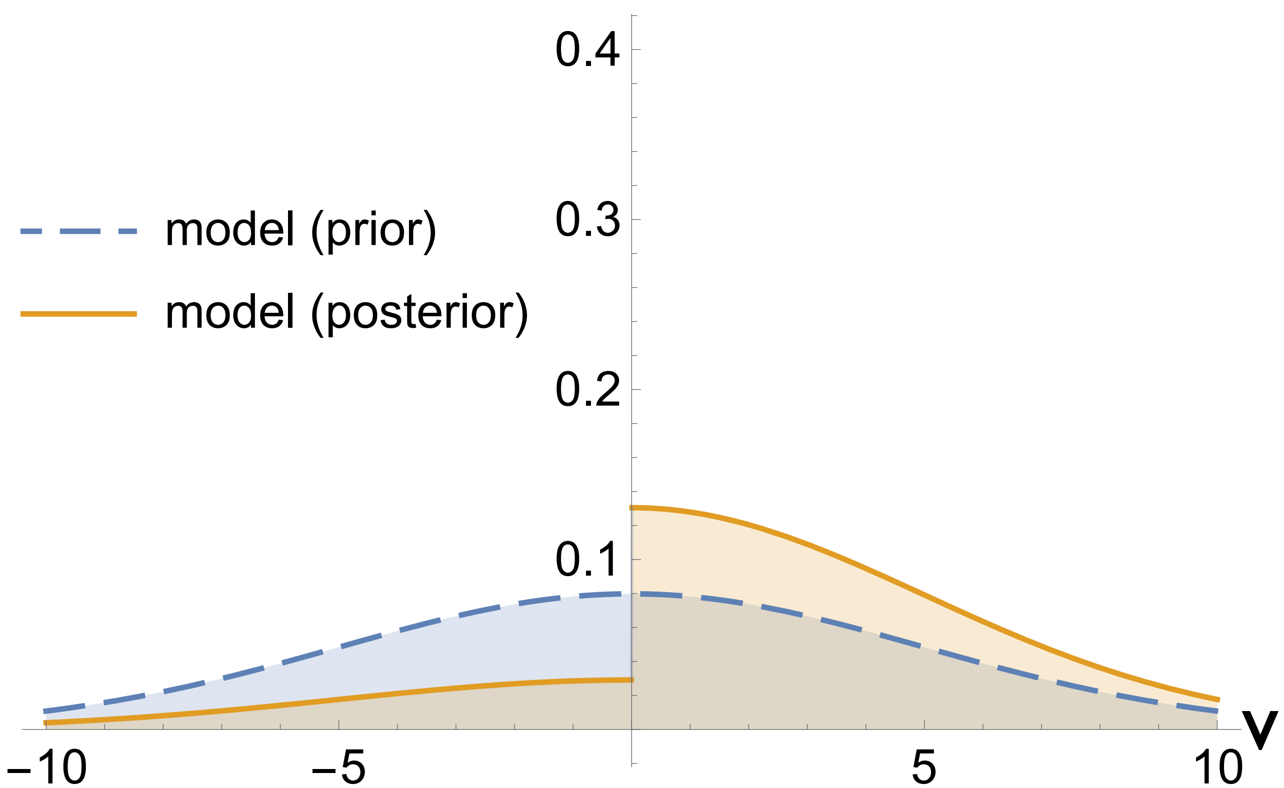
$KL[q_\theta(z)||p(z|x)]$ may fail to be differentiable wrt. θ.

```
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.)
```



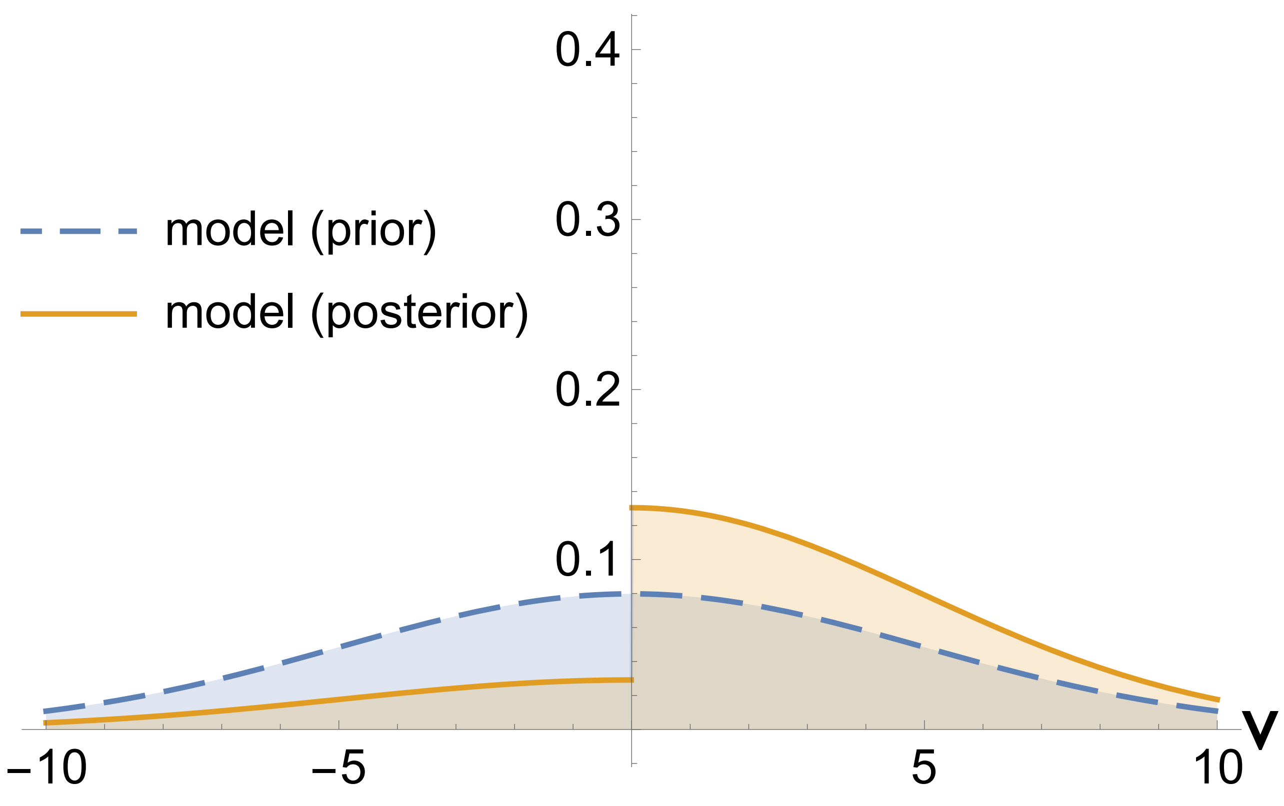
```
def p(): // model_1
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):  pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:  pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

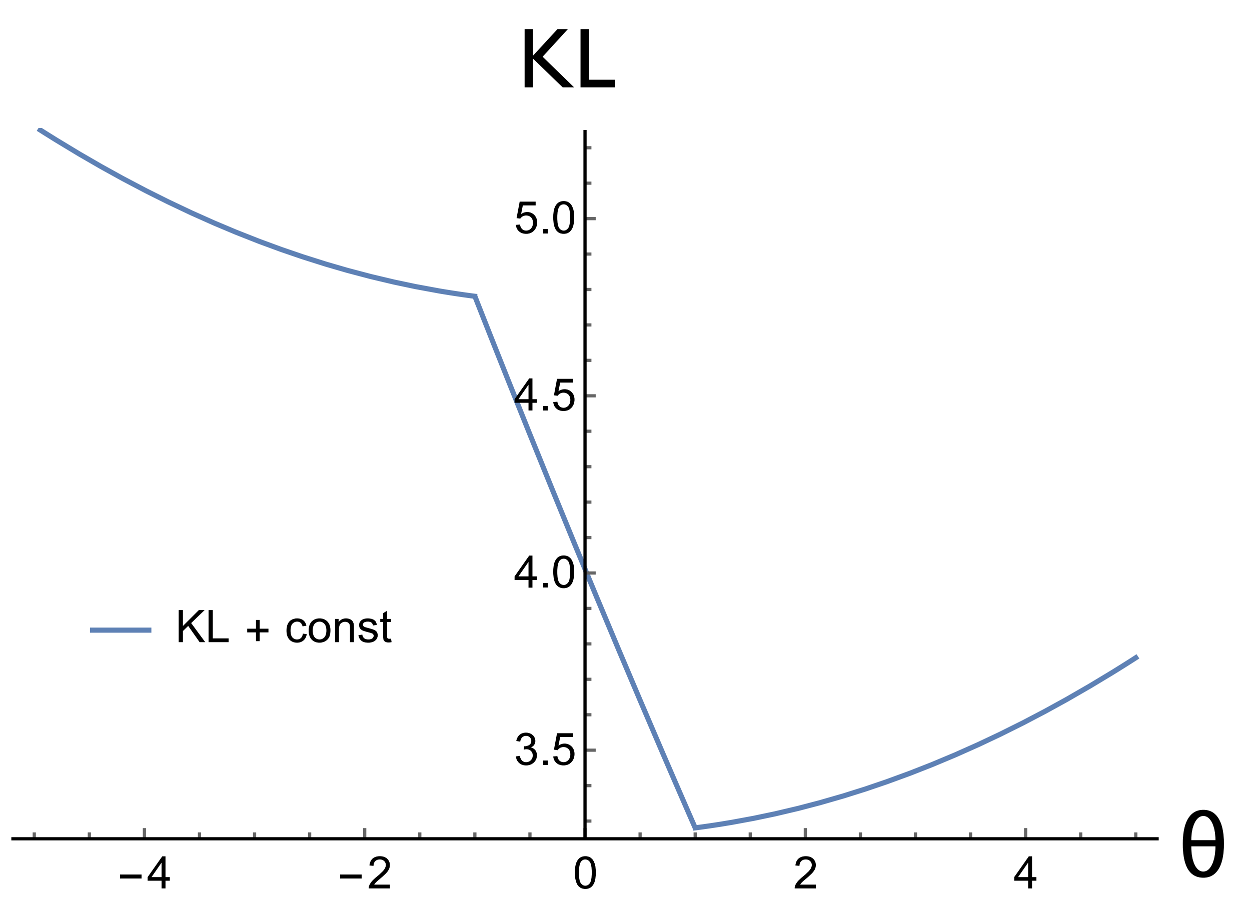
```
def qθ(): // guide_1'
  θ = pyro.param("θ", 4.)
  v = pyro.sample("v", Uniform(θ -1., θ +1.))
```



```
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.)
```

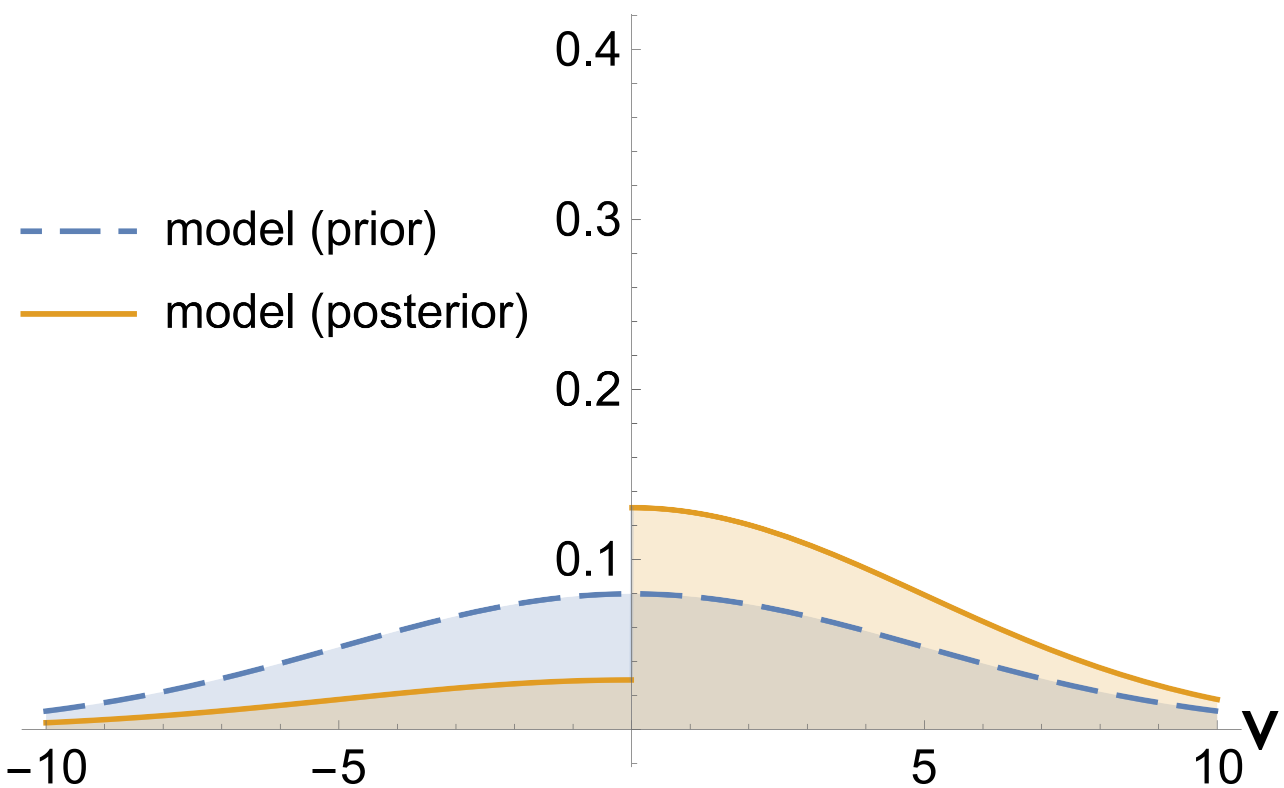
```
def qθ(): // guide_1'
  θ = pyro.param("θ", 4.)
  v = pyro.sample("v", Uniform(θ -1., θ +1.))
```

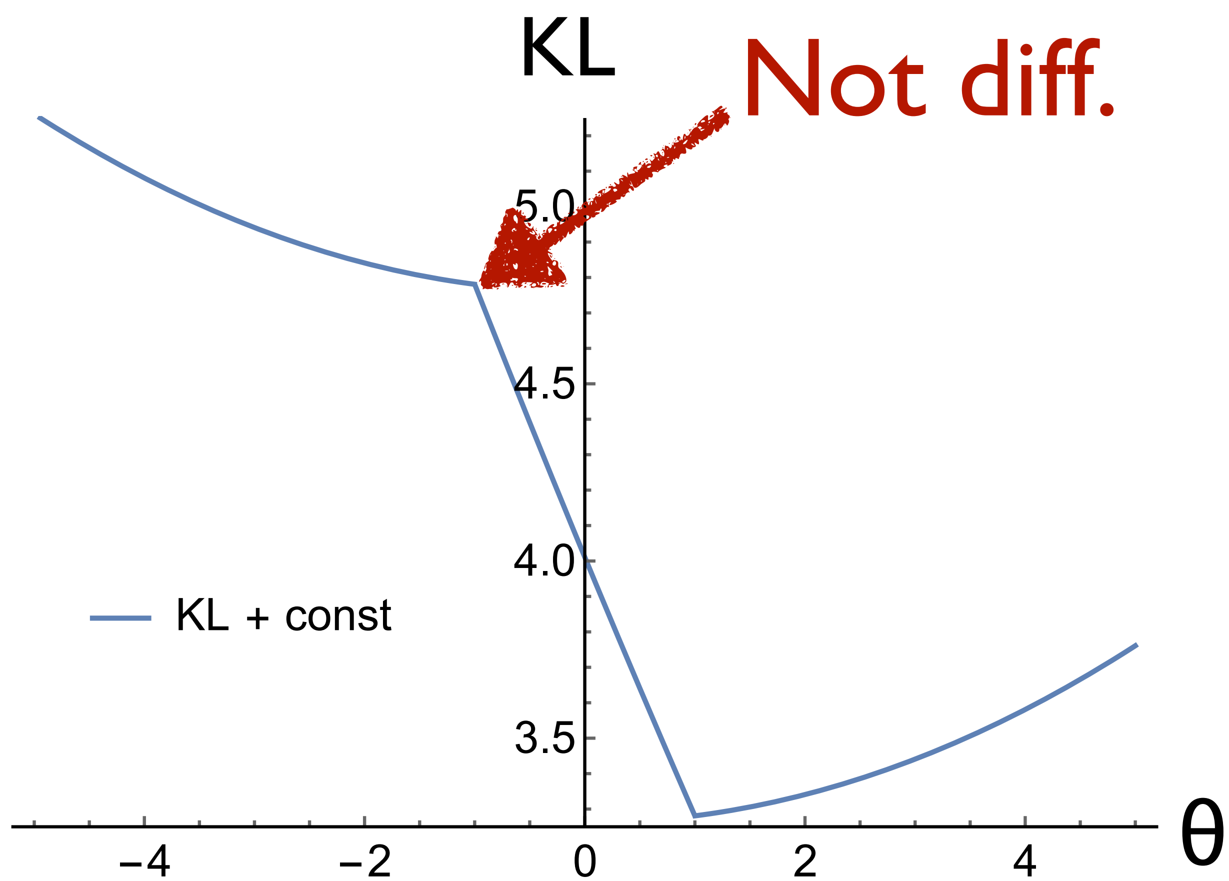




```
def p(): // model_1
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):  pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:  pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

```
def qθ(): // 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,

$$\nabla_\theta \mathrm{KL}[q_\theta(z)||p(z|x)] \neq \mathbb{E}[\ \widehat{\nabla_\theta \mathrm{KL}[q_\theta(z)||p(z|x)]}\ ],$$

when we expect equality.

Score estimator

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

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

where z_0 is sampled from q_θ.

Score estimator

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

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

where z_0 is sampled from q_θ.

Thm: $\nabla_\theta KL[q_\theta(z) || p(z|x)] = \mathbb{E}[\widehat{\nabla_\theta \, KL[q_\theta(z) || p(z|x)]}]$

Score estimator

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

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

where z_0 is sampled from q_θ.

Thm: $\nabla_\theta KL[q_\theta(z)||p(z|x)] = \mathbb{E}[\widehat{\nabla_\theta \, KL[q_\theta(z)||p(z|x)]}]$

if some requirements are met.

Proof of the theorem

$\nabla_\theta \, KL[q_\theta(z)||p(z|x)]$

$= \nabla_\theta \int dz \; (q_\theta(z) \times \log \, (q_\theta(z)/p(z|x)))$

$= \int dz \; (\nabla_\theta(q_\theta(z) \times \log \, (q_\theta(z)/p(z|x))))$

…

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

Proof of the theorem

Interchange of integration and differentiation. Might fail.

$\nabla_\theta \, KL[q_\theta(z) || p(z|x)]$

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

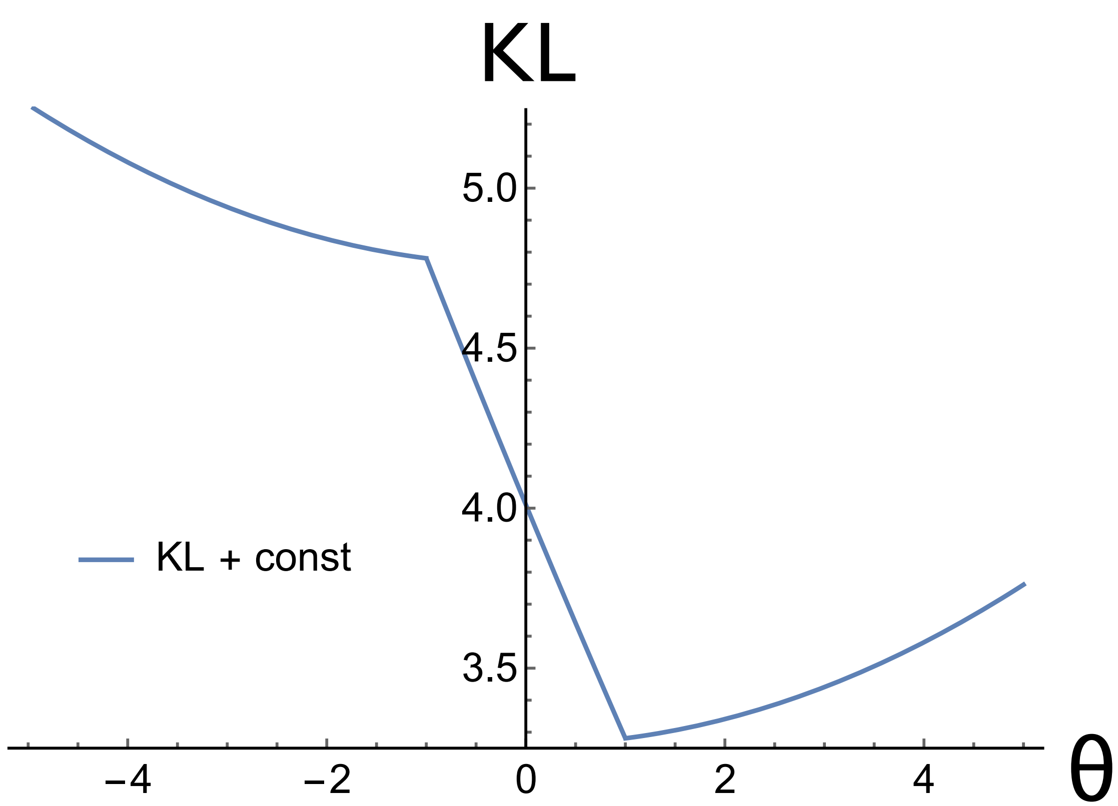
$= \int dz \, (\nabla_\theta(q_\theta(z) \times \log (q_\theta(z)/p(z|x))))$

…

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

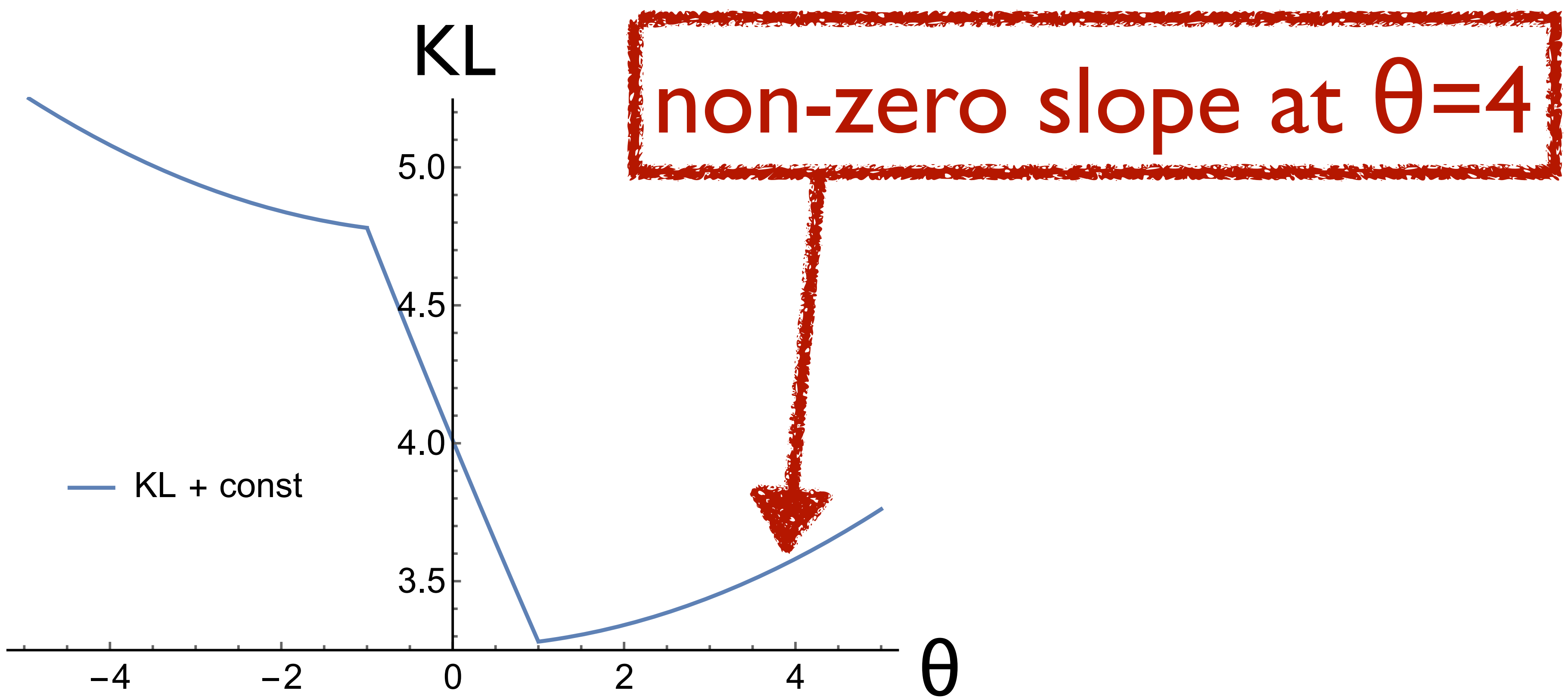
```
def p(): // model_1
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):  pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:  pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

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



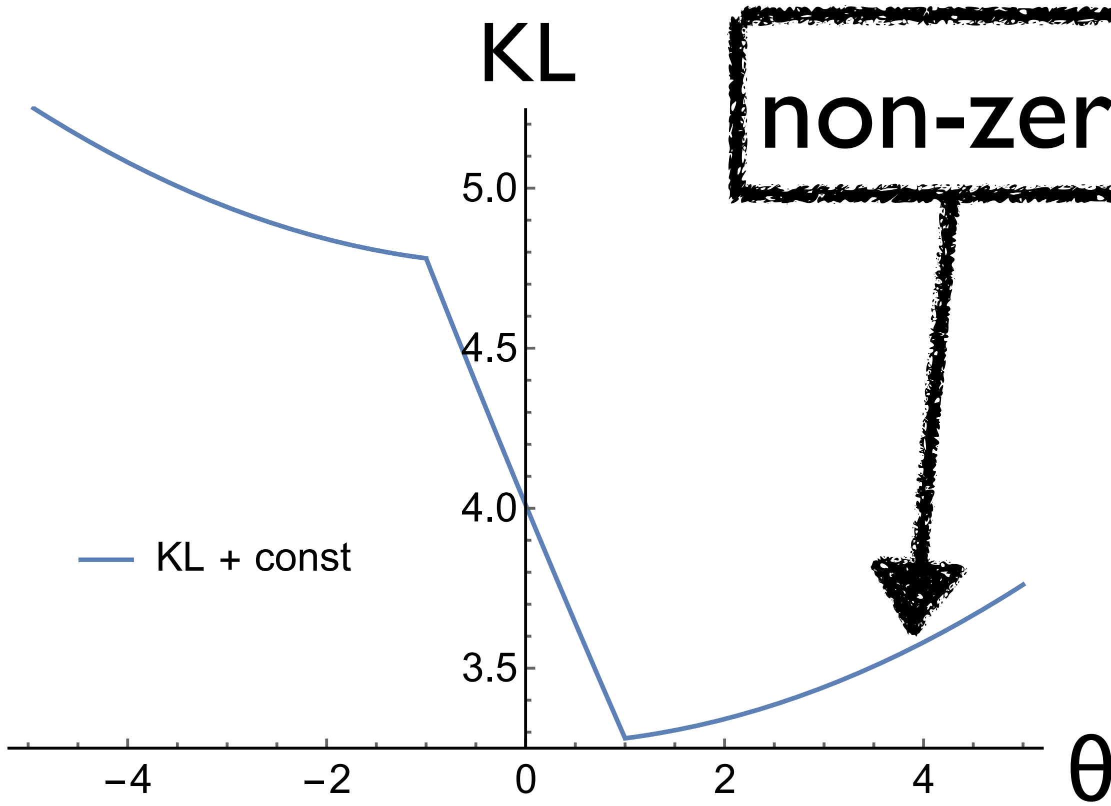
```
def p(): // model_1
  v = pyro.sample("v", Normal(0., 5.))
  if (v > 0):  pyro.sample("obs", Normal(1., 1.), obs=0.)
  else:  pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

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

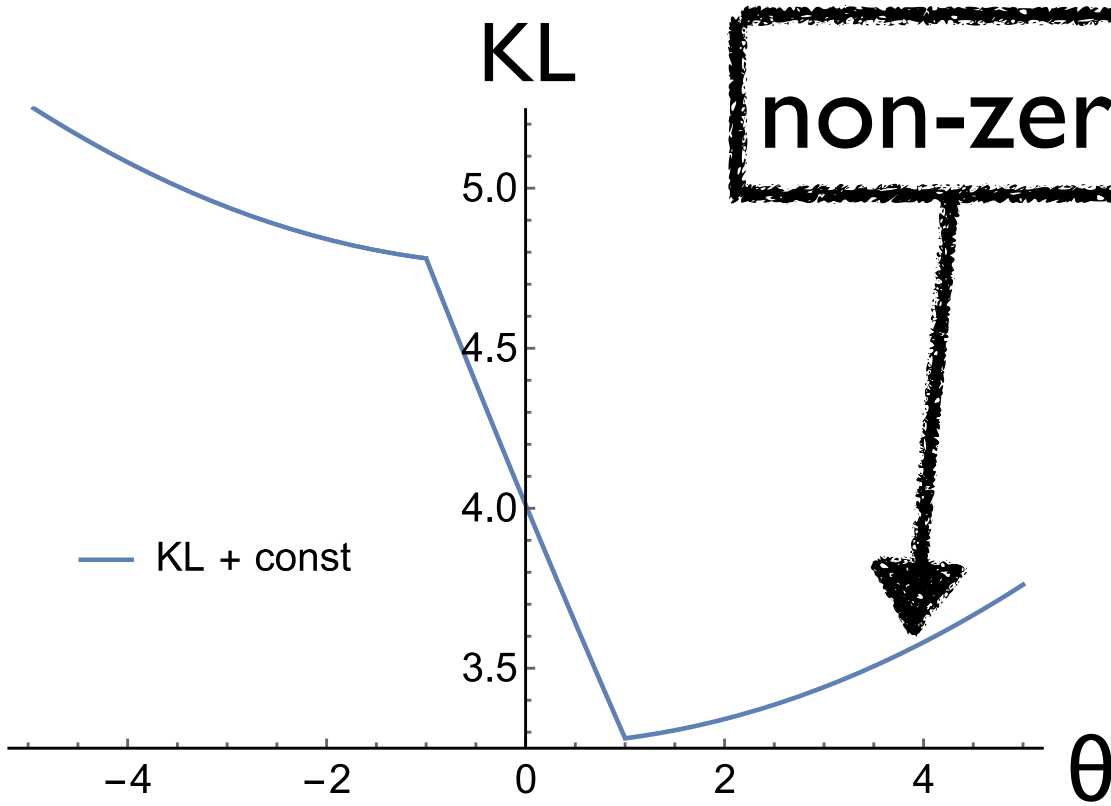


non-zero slope at θ=4

$$\nabla_\theta KL[q_\theta(z)||p(z|x)]_{\theta=4}$$

```
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.))
```



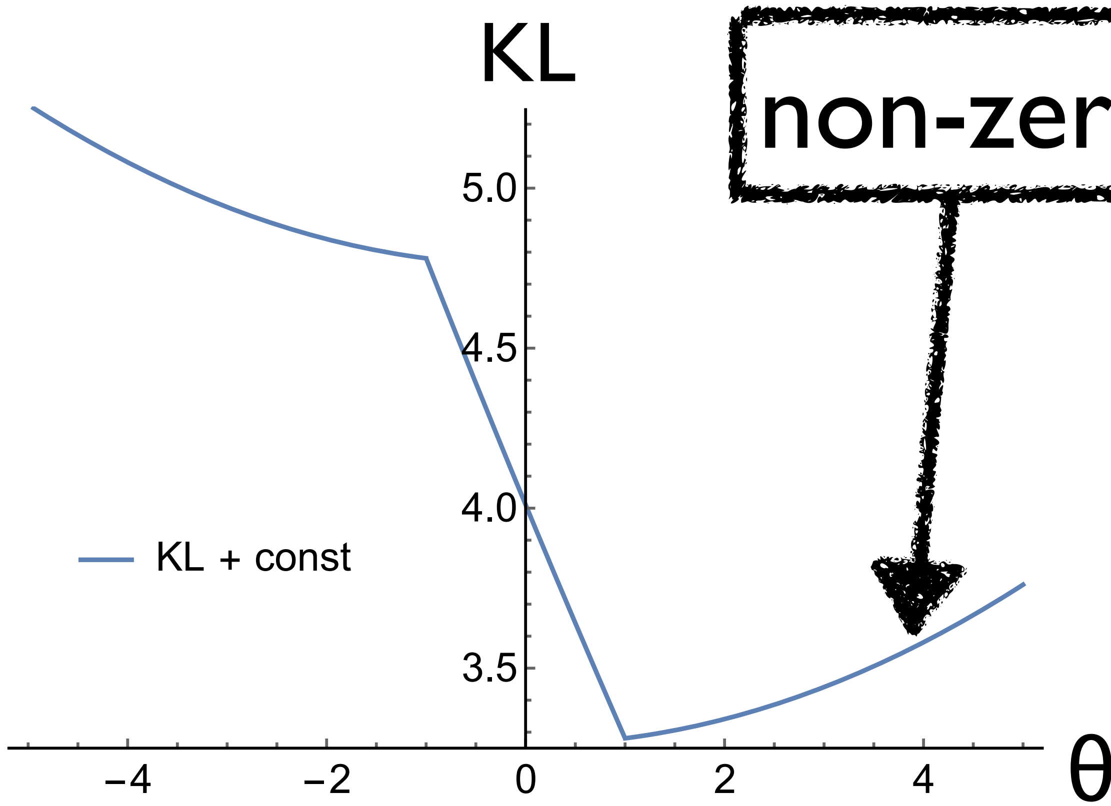
non-zero slope at θ=4

$\nabla_\theta KL[q_\theta(z)||p(z|x)]_{\theta=4}$

$= (\nabla_\theta \log q_\theta(z_0))_{\theta=4}\ldots$

```
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.))
```



non-zero slope at θ=4

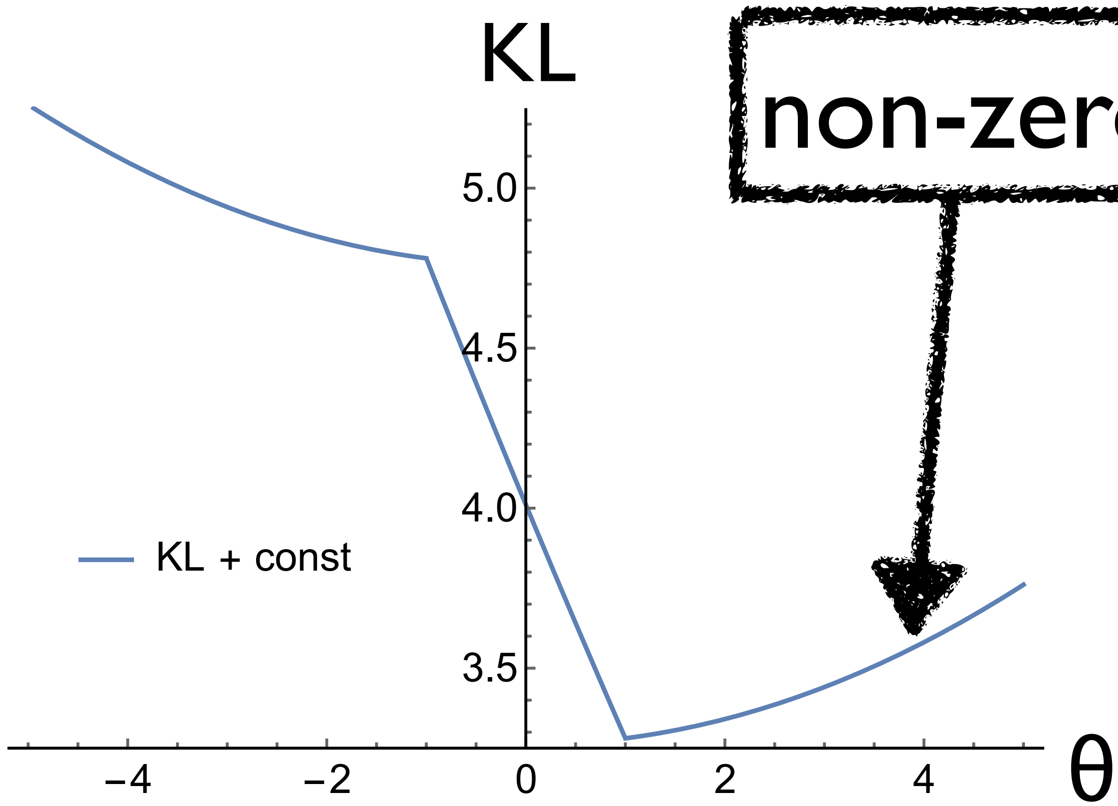
$\nabla_\theta \mathrm{KL}[q_\theta(z)||p(z|x)]_{\theta=4}$

$= (\nabla_\theta \log q_\theta(z_0))_{\theta=4} \ldots$

$= (\nabla_\theta \log 0.5) \ldots$

```
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.))
```



non-zero slope at θ=4

$$\nabla_\theta \mathrm{KL}[q_\theta(z)||p(z|x)]_{\theta=4}$$
$$= (\nabla_\theta \log q_\theta(z_0))_{\theta=4}\ldots$$
$$= (\nabla_\theta \log 0.5)\ldots$$
$$= 0$$

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 $\nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots$

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 $\nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots$

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

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

```
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", 0.)
   v = pyro.sample("v", Normal(θ, 1.))
```

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

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

```
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", 0.)
   v = pyro.sample("v", Normal(θ, 1.))
```

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 $\nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots$

Sufficient condition

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

Sufficient condition

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

Sufficient condition

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

1. μ, σ are continuously differentiable wrt. θ.
2. $|\mu'(z)| \leq \exp(f(|z|))$ for affine f.

Sufficient condition

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

1. μ, σ are continuously differentiable wrt. θ.
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 qθ(): // guide_1
  θ = pyro.param("θ", 0.)
  v = pyro.sample("v", Normal(θ, 1.))
```

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

1. μ, σ are continuously differentiable wrt. θ.
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 qθ(): // guide_1
   θ = pyro.param("θ", 0.)
   v = pyro.sample("v", Normal(θ, 1.))
```

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

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 qθ(): // guide_1
   θ = pyro.param("θ", 0.)
   v = pyro.sample("v", Normal(θ, 1.))
```

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

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

3. $\exp(g(|z|)) \le |\sigma'(z)| \le \exp(h(|z|))$ 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 qθ(): // guide_1
   θ = pyro.param("θ", 0.)
   v = pyro.sample("v", Normal(θ, 1.))
```

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

✓ 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.

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

1. μ, σ are continuously differentiable wrt. θ.
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_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 $q_\theta(z)$.

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

1. μ, σ are continuously differentiable wrt. θ.
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_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 $q_\theta(z)$.

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

1. μ, σ are continuously differentiable wrt. θ.
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_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 $q_\theta(z)$.

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

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

✓ 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_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 $q_\theta(z)$.

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

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

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

$\boxed{\text{no}}$ 3. $\exp(g(|z|)) \le |\sigma'(z)| \le \exp(h(|z|))$ for affine g,h.

Sufficient condition

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

$\boxed{\begin{array}{ll} 1. & \text{KL not integrable.} \\ 2. & \text{KL not differentiable.} \\ 3. & \text{Biased due to } \nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots \end{array}}$

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

1. μ, σ are continuously differentiable wrt. θ.
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 $\nabla_\theta \int \ldots \neq \int \nabla_\theta \ldots$

Because … becomes a good fn (C^1 & dominated).

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

μ, σ - mean, standard deviation in $q_\theta(z)$.

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

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

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

- Handles features of Python/PyTorch/Pyro, such as tensor broadcasting, but not all of them.

Name	Corresponding probabilistic model	LoC	Total #				Total dimension		
			`for`	`plate`	`sample`	`score`	`sample`	`score`	θ
`br`	Bayesian regression	27	0	1	10	1	10	170	9
`csis`	Compiled sequential importance sampling	31	0	0	2	2	2	2	480
`lda`	Latent Dirichlet allocation (LDA)	76	0	5	8	1	21008	64000	121400
`vae`	Variational autoencoder (VAE)	91	0	2	2	1	25600	200704	353600
`sgdef`	Sparse gamma deep exponential family	94	0	8	12	1	231280	1310720	231280
`dmm`	Deep Markov model	246	3	2	2	1	640000	281600	594000
`ssvae`	Semi-supervised VAE	349	0	2	4	1	24000	156800	844000
`air`	Attend-infer-repeat (AIR)	410	2	2	6	1	20736	160000	6040859

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 θ in guide.

Analysed 8 representative Pyro programs from Pyro webpage.

Name	Valid?	Time
`br`	x	0.006
`csis`	o	0.007
`lda`	x	0.014
`vae`	o	0.005
`sgdef`	o	0.070
`dmm`	o	0.536
`ssvae`	o	0.013
`air`	o	4.093

Uniform in p
Normal in qθ

Name	Valid?	Time
`br`	x	0.006
`csis`	o	0.007
`lda`	x	0.014
`vae`	o	0.005
`sgdef`	o	0.070
`dmm`	o	0.536
`ssvae`	o	0.013
`air`	o	4.093

Uniform in p
Normal in q_θ

Name	Valid?	Time
br	x	0.006
csis	o	0.007
lda	x	0.014
vae	o	0.005
sgdef	o	0.070
dmm	o	0.536
ssvae	o	0.013
air	o	4.093

Dirichlet in p
Delta in q_θ

Reference

The details can be found in our archive paper:

Towards Verified Stochastic Variational Inference for Probabilistic Programs. https://arxiv.org/abs/1907.08827