

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)

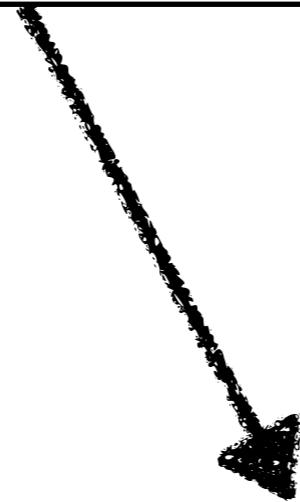
What are ideal programming language and system
for ML?

**Support auto. differentiation and deep learning
— TensorFlow, PyTorch**



**What are ideal programming language and system
for ML?**

Support auto. differentiation and deep learning
— TensorFlow, PyTorch

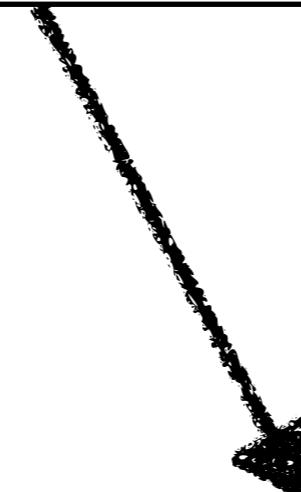


What are ideal programming language and system
for ML?



Support deep probabilistic programming
— Pyro, Edward, ProbTorch, PyProb

Support auto. differentiation and deep learning
— TensorFlow, PyTorch



What are ideal programming language and system
for ML?



Support deep probabilistic programming
— Pyro, Edward, ProbTorch, PyProb

&

Help use learning/inference
algorithms (such as SVI) correctly.

High-level message I

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

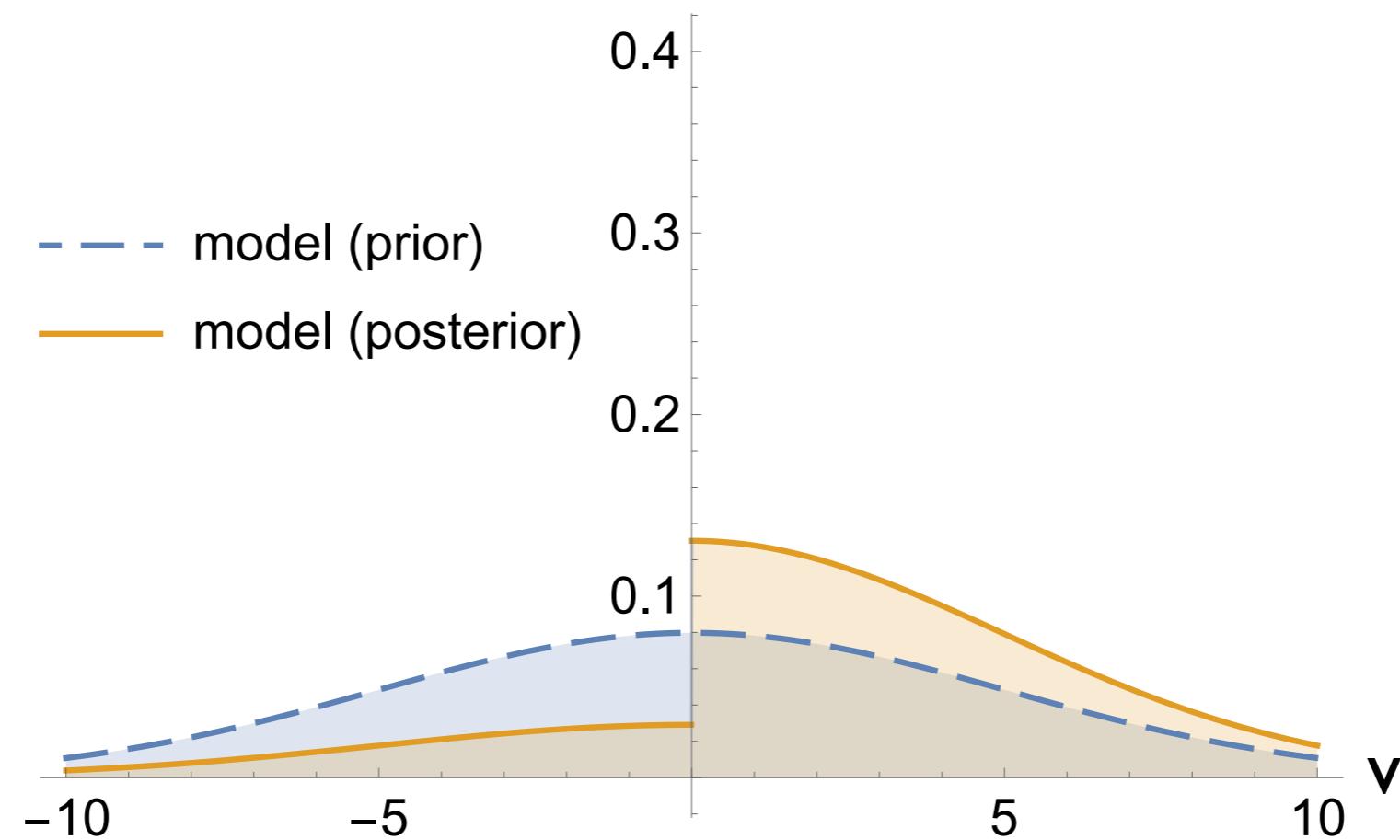
```
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

```
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.)
```

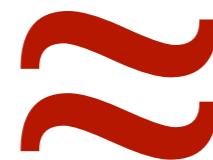
```
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)
```

```
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.)
```

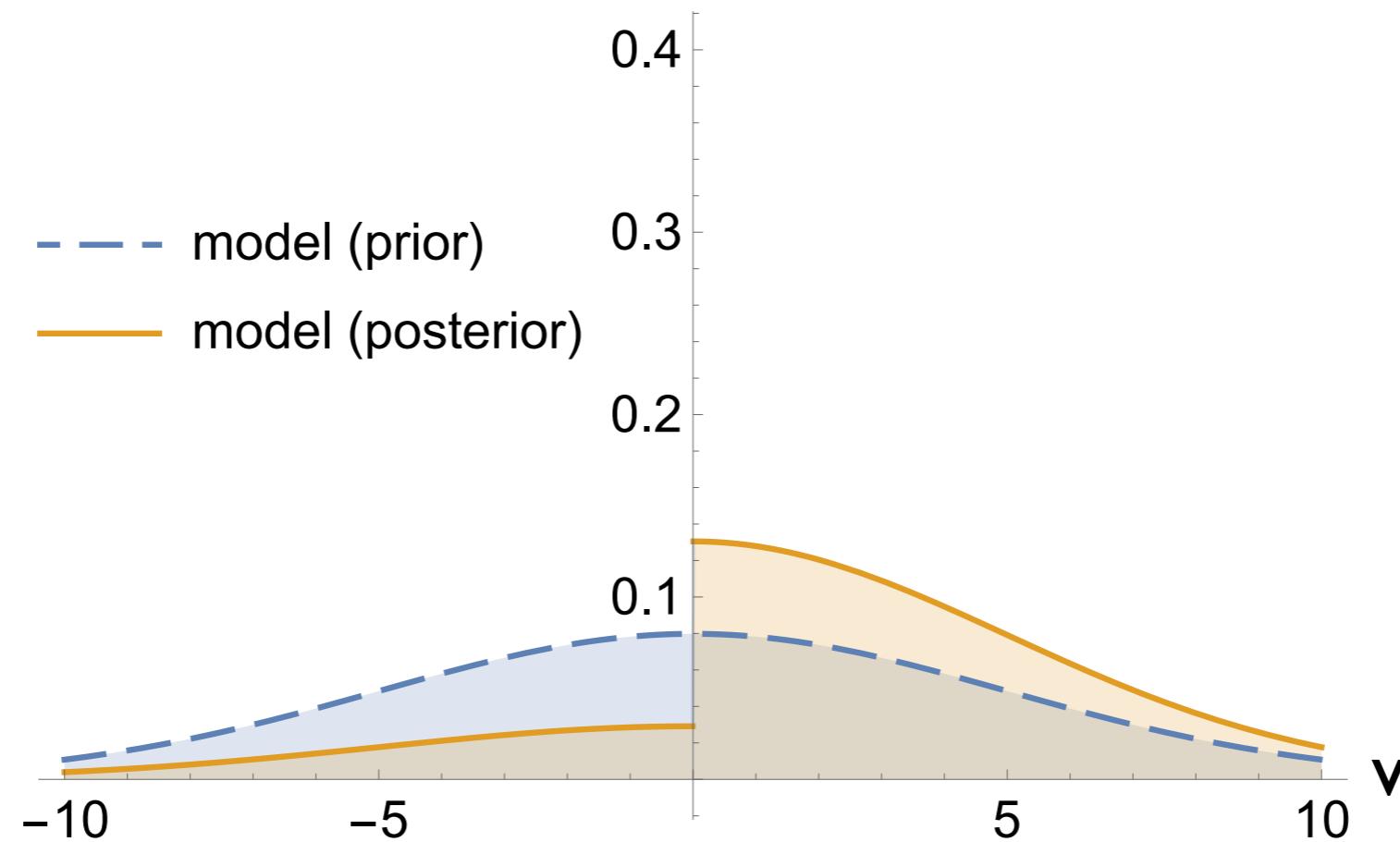
```
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)
```



```
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.)
```



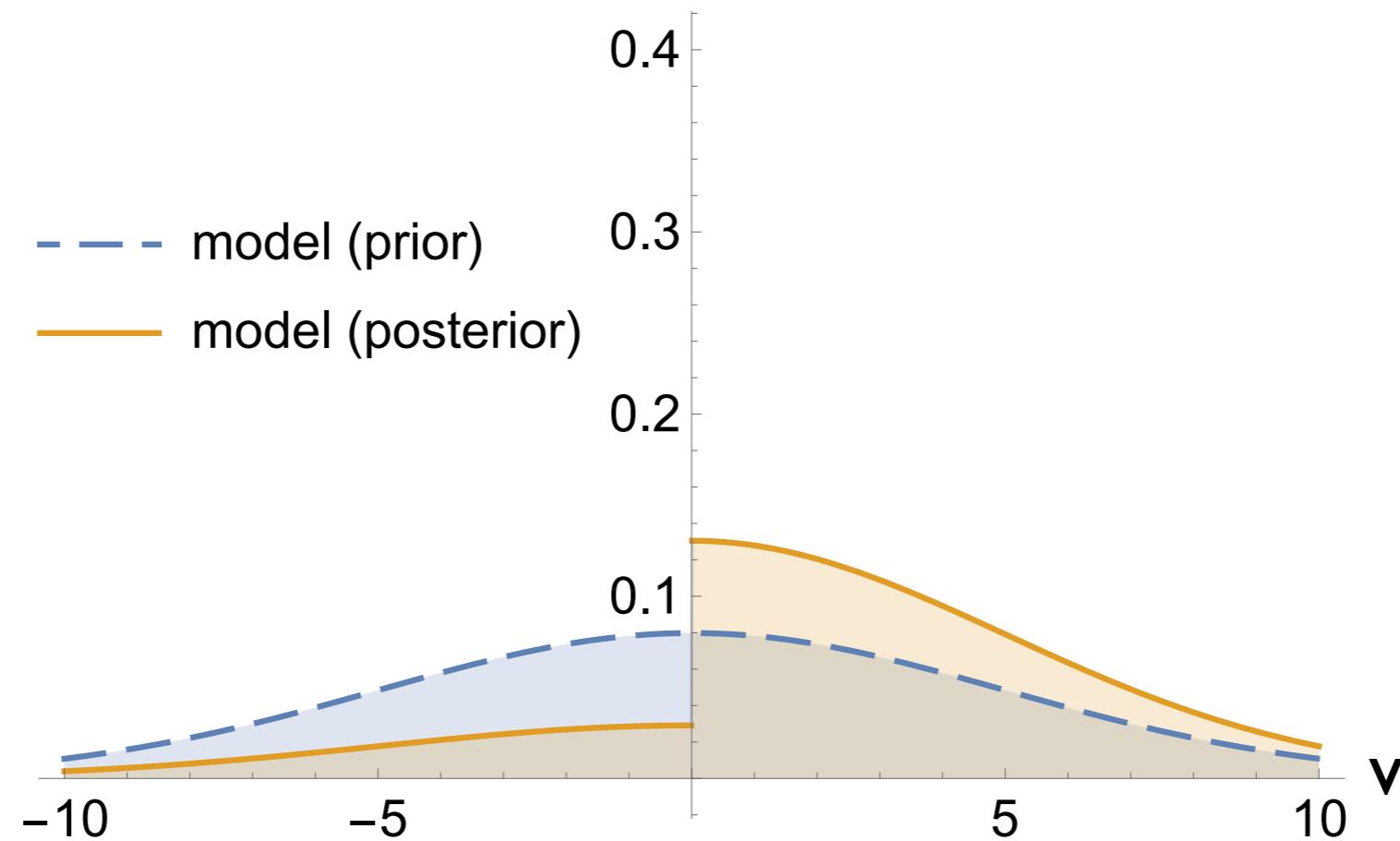
```
def qθ(): // guide_1
    θ = pyro.param("θ", 0.)
    v = pyro.sample("v", Normal(θ, 1.))
```



```
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.)

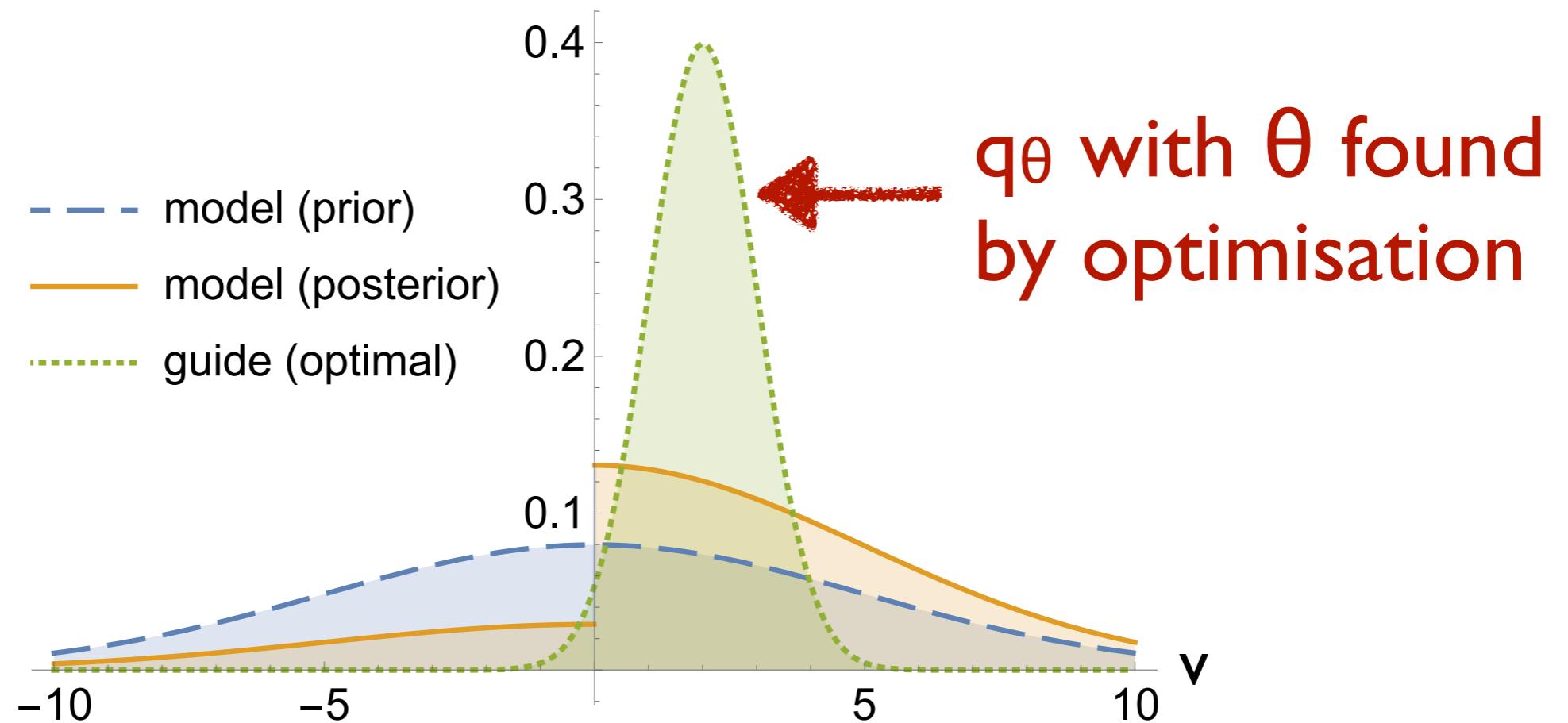
```

\approx

```

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

```



Typical optimisation objective:

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

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

Typical optimisation objective:

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

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

Typical optimisation objective:

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

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

Typical optimisation objective:

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

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:

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

Typical optimisation objective:

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

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:

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

Issue I: Undefined KL

Typical optimisation objective:

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

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:

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

Issue I: Undefined KL

Typical optimisation objective:

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

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:

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

Issue I: Undefined KL

Typical optimisation objective:

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

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:

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

Issue 1: Undefined KL

Typical optimisation objective:

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

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:

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

Issue 2: Non-differentiable KL

Issue 1: Undefined KL

Typical optimisation objective:

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

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:

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

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 I: Undefined KL

$$\begin{aligned} \text{KL}[q_\theta || p] &= \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \\ &= \int dz (q_\theta(z) \log (q_\theta(z)/p(z|x))) \end{aligned}$$

Issue I: Undefined KL

$$\begin{aligned} \text{KL}[q_\theta || p] &= \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \\ &= \int dz (q_\theta(z) \log (q_\theta(z)/p(z|x))) \end{aligned}$$

Two reasons for being undefined.

Issue I: Undefined KL

$$\begin{aligned} \text{KL}[q_\theta \parallel p] &= \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \\ &= \int dz (q_\theta(z) \log (q_\theta(z)/p(z|x))) \end{aligned}$$

Two reasons for being undefined.

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

Issue I: Undefined KL

$$\begin{aligned} \text{KL}[q_\theta \parallel p] &= \mathbb{E}_{q_\theta(z)}[\log (q_\theta(z)/p(z|x))] \\ &= \int dz (q_\theta(z) \log (q_\theta(z)/p(z|x))) \end{aligned}$$

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))
```

$\text{KL}[q_{\theta}(z) \parallel p(z|x)]$ 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(0, 0.05))
```

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

$\text{KL}[q_\theta(z) \parallel p(z|x)]$ 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  
    ...  
    sigma = pyro.sample("sigma", Normal(0, 0.05))
```

Uniform(0., 10.)

$\text{KL}[q_\theta(z) \parallel p(z|x)]$ 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  
    ...  
    sigma = pyro.sample("sigma", Uniform(0., 10.))  
    ...  
    sigma = pyro.sample("sigma", Normal(0, 0.05))
```

Not integrable.
~~KL[q_θ(z)||p(z|x)] undefined. Bad Integrand!~~

[Q] Fix it.

Bayesian regression from Pyro

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

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

```
def qθ(...): // guide_br  
...  
sigma = pyro.sample("sigma", Uniform(0., 10.))
```

Not integrable.
~~KL[q_θ(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=...)
```

Normal(0., 5.0)
~~Uniform(0., 10.)~~
abs(sigma)

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

Not integrable.
~~KL[q_θ(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=...)
```

Normal(0., 5.0)
~~Uniform(0., 10.)~~
abs(sigma)

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

Not integrable.
~~KL[q_θ(z)||p(z|x)] undefined. Bad Integrand!~~

[Q] Fix it.

Bayesian regression fr

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

...

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

...

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

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

abs(sigma)

```
def q_theta(...): // guide_br
```

...

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

Not integrable.

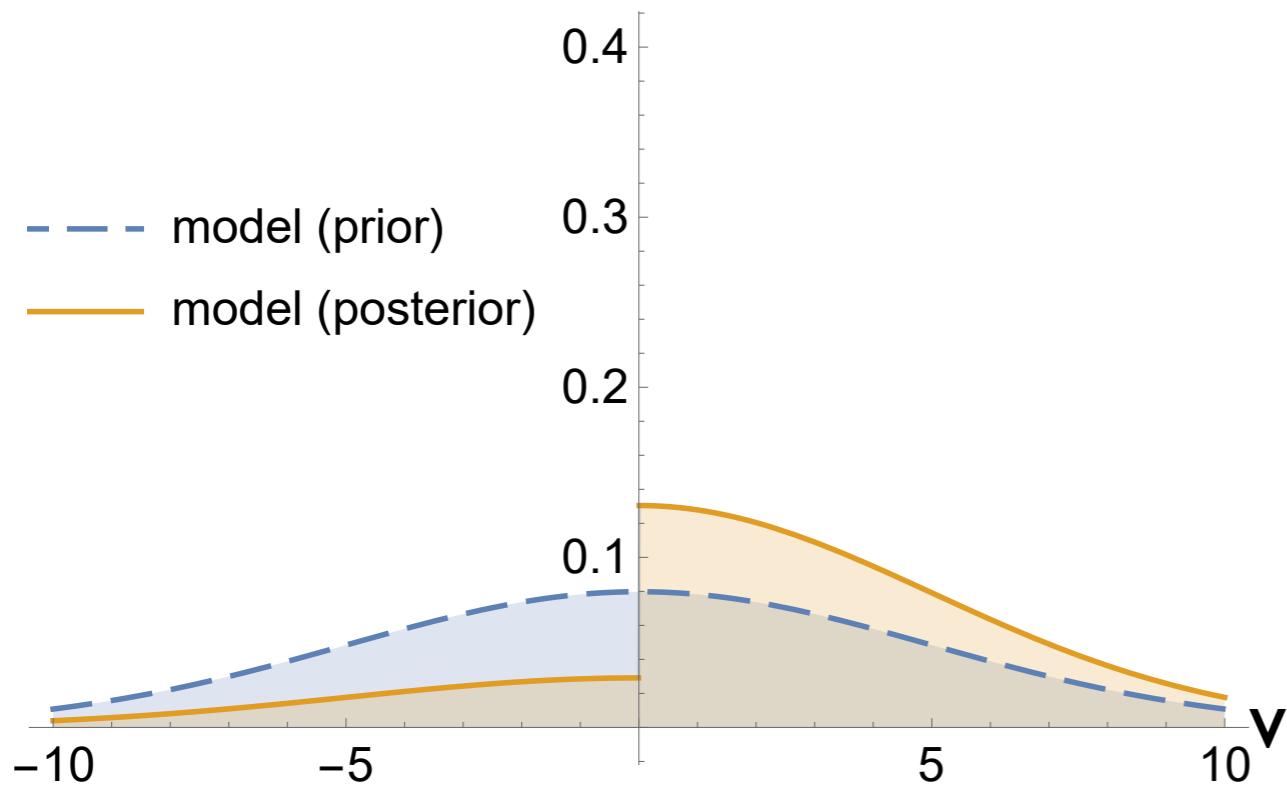
KL[q_θ(z)||p(z|x)] undefined. Bad Integrand!

[Q] Fix it.

Issue 2: Non-differentiable KL

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



```

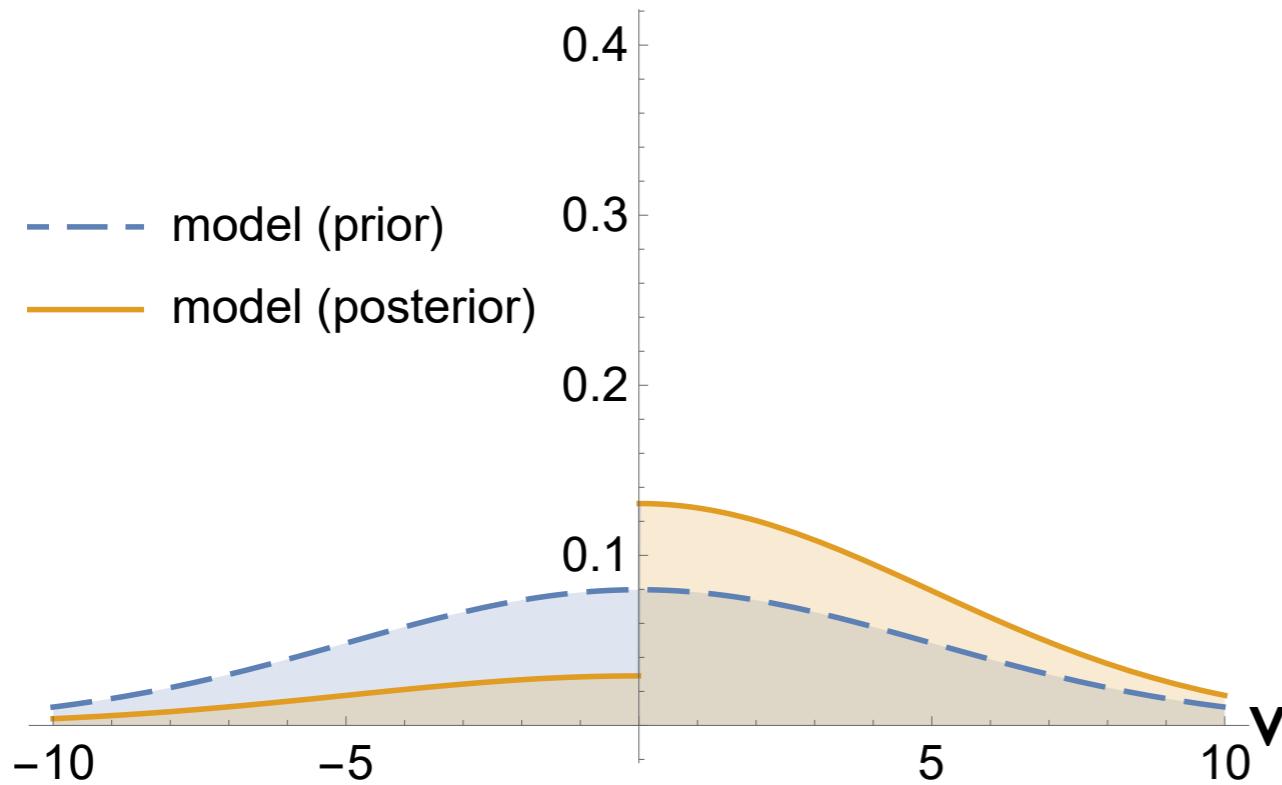
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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

```



```

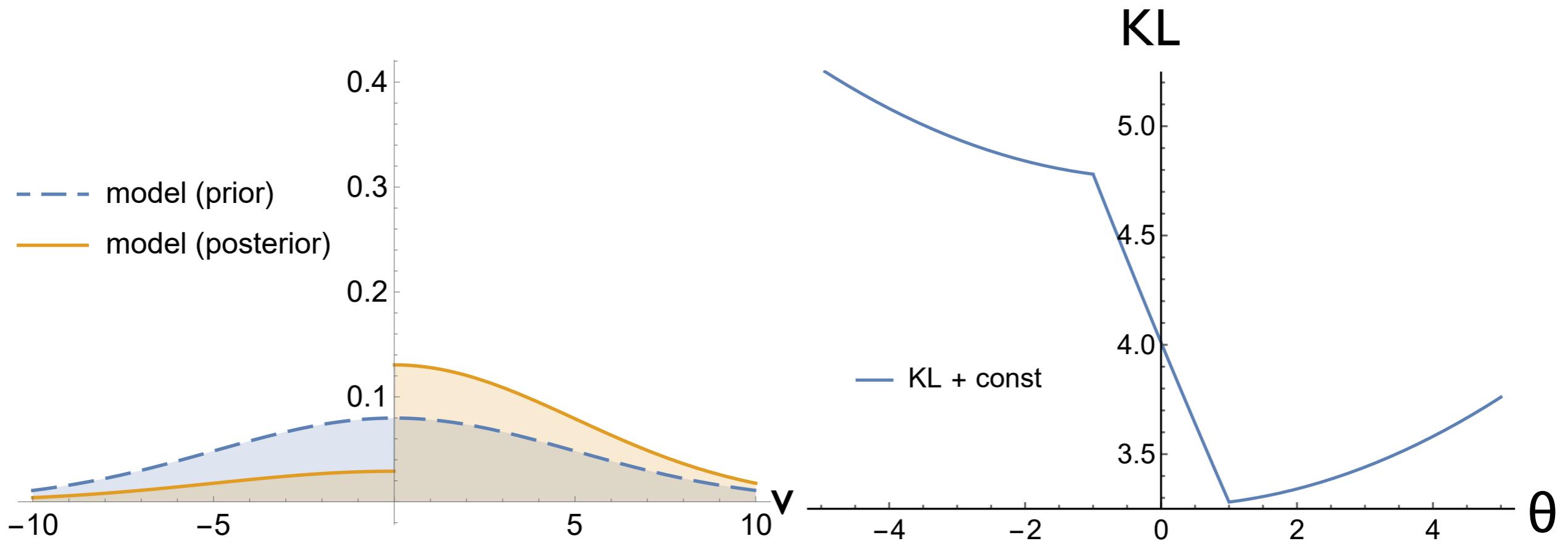
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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

```



```

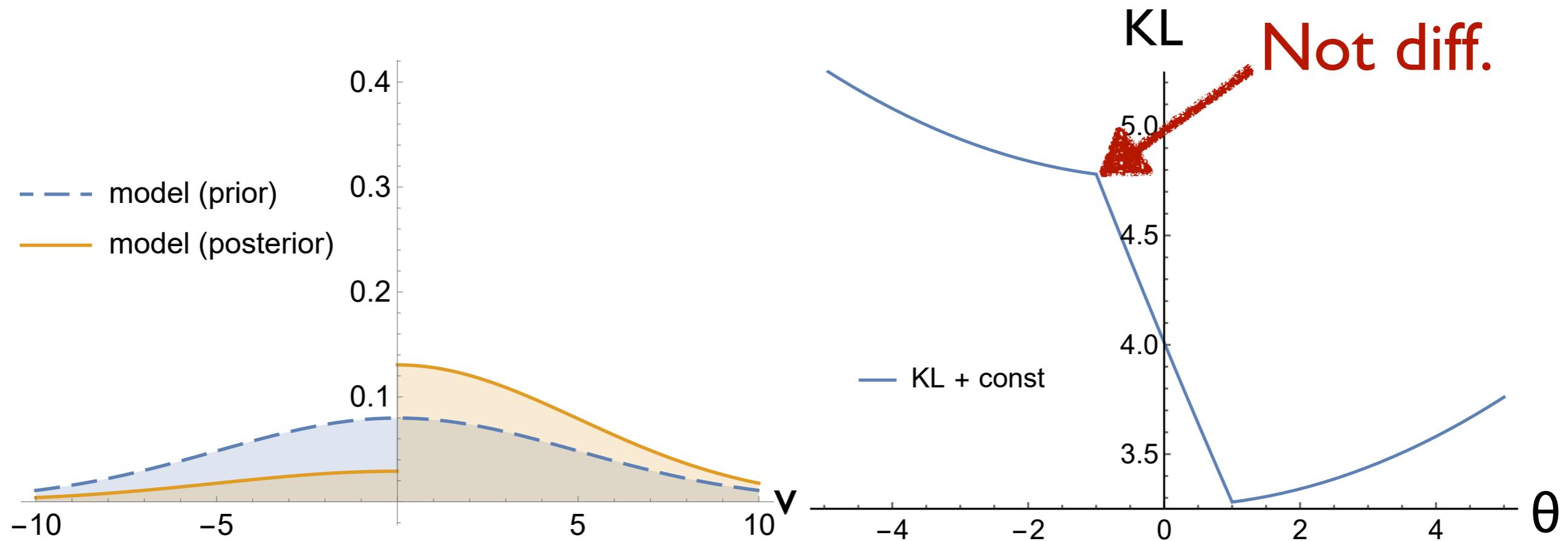
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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,

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

when we expect equality.

Score estimator

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

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

where z_0 is sampled from q_{θ} .

Score estimator

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

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

where z_0 is sampled from q_{θ} .

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

Score estimator

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

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

where z_0 is sampled from q_{θ} .

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

if some requirements are met.

Proof of the theorem

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

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

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

...

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

Proof of the theorem

Interchange of integration and differentiation. Might fail.

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

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

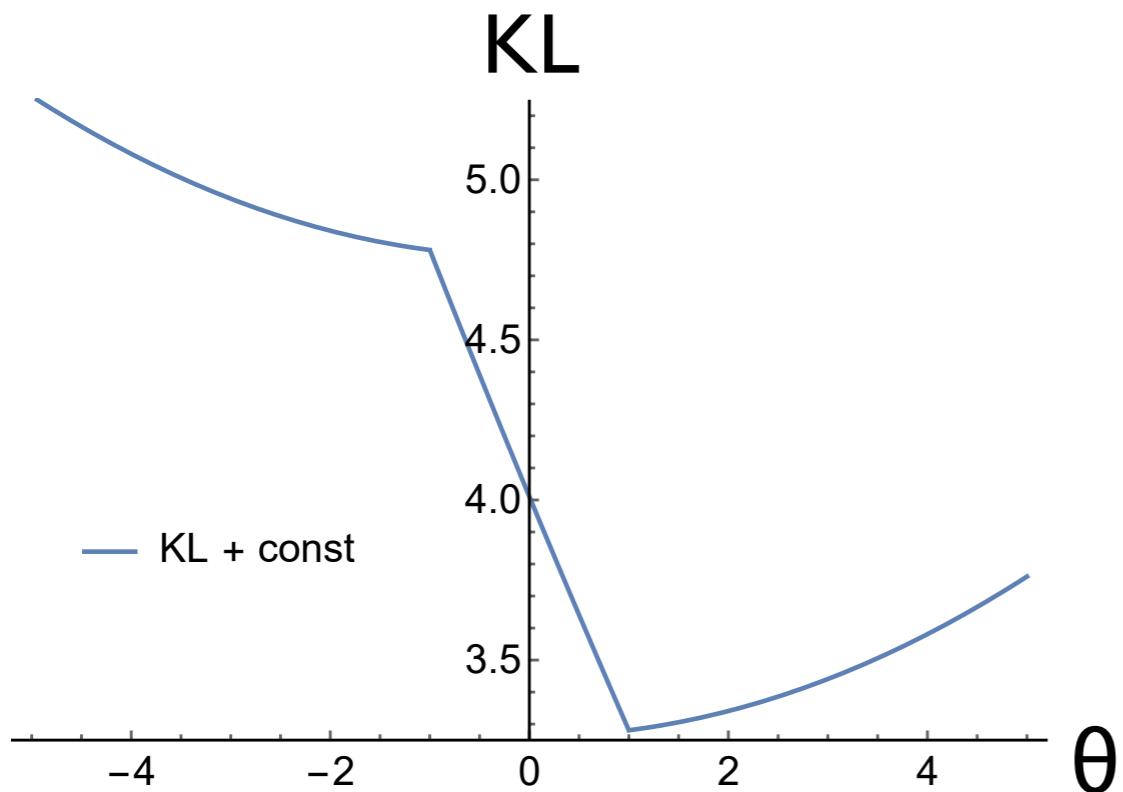
$$= \int dz (\nabla_{\theta}(q_{\theta}(z)) \times \log (q_{\theta}(z)/p(z|x))))$$

...

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

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



```

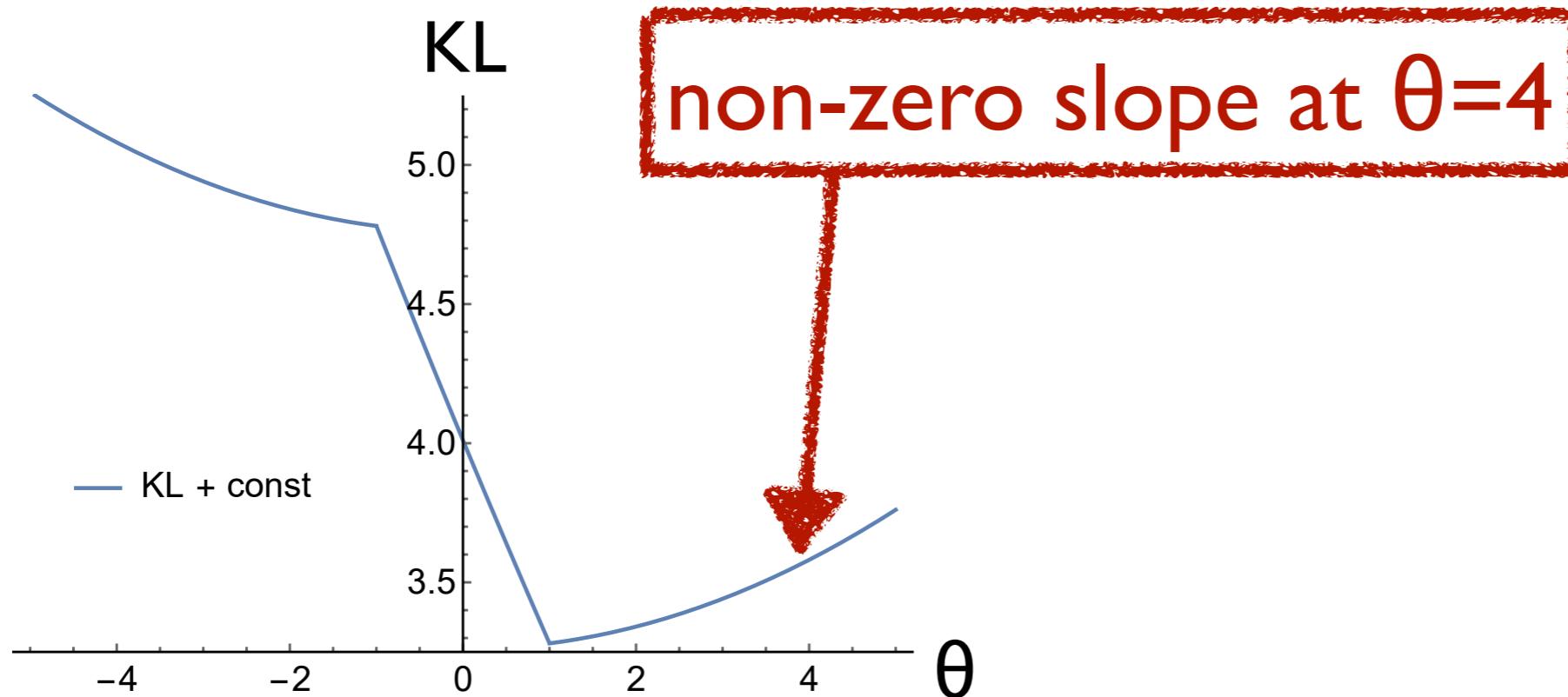
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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

```



```

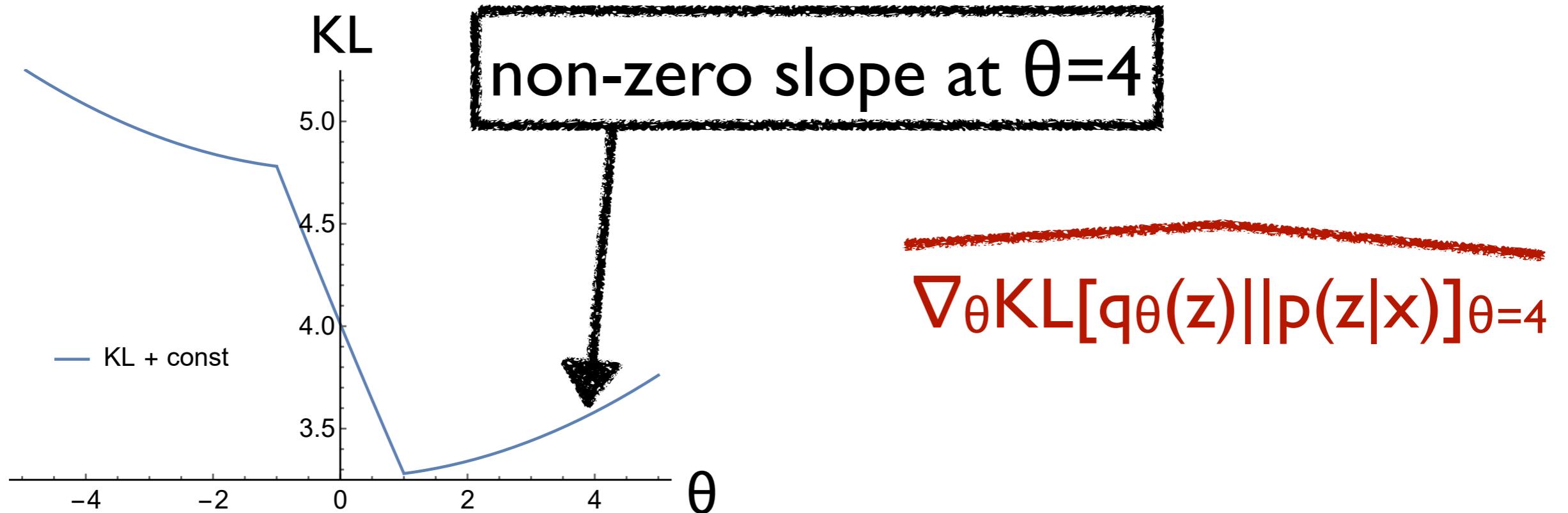
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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

```



```

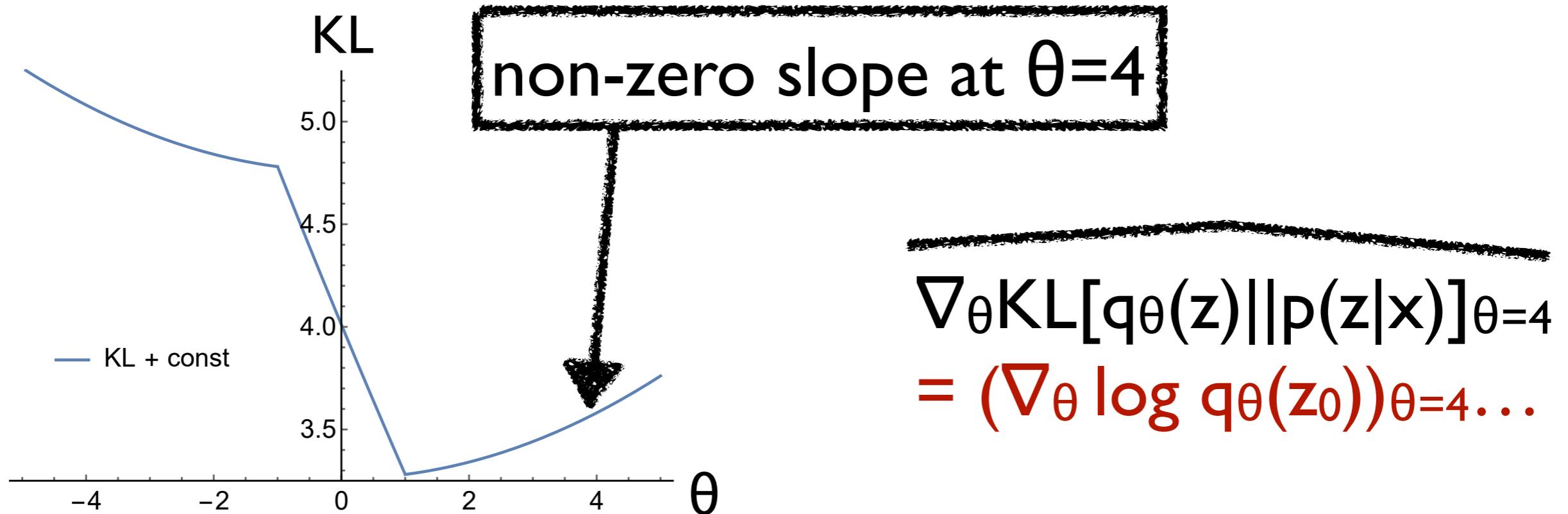
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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

```



```

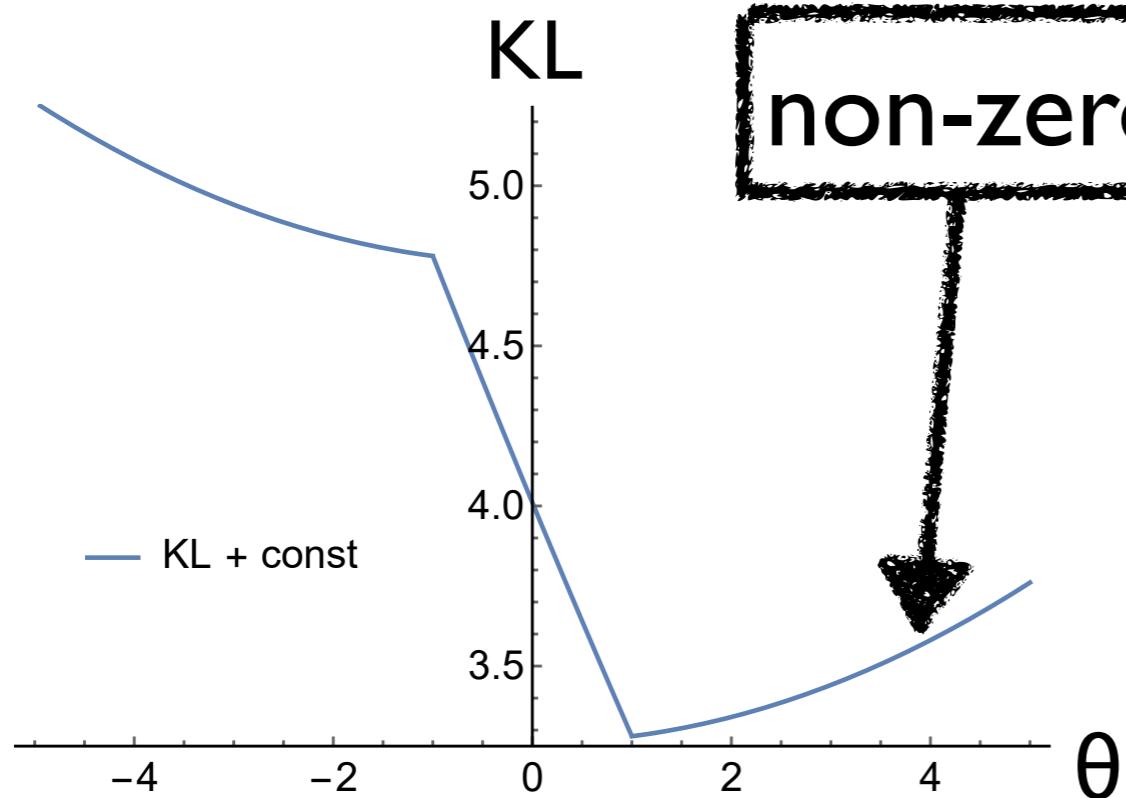
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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

```



$$\begin{aligned}
 \nabla_{\theta} \text{KL}[q_{\theta}(z) || p(z|x)]_{\theta=4} \\
 &= (\nabla_{\theta} \log q_{\theta}(z_0))_{\theta=4} \dots \\
 &= (\nabla_{\theta} \log 0.5) \dots
 \end{aligned}$$

```

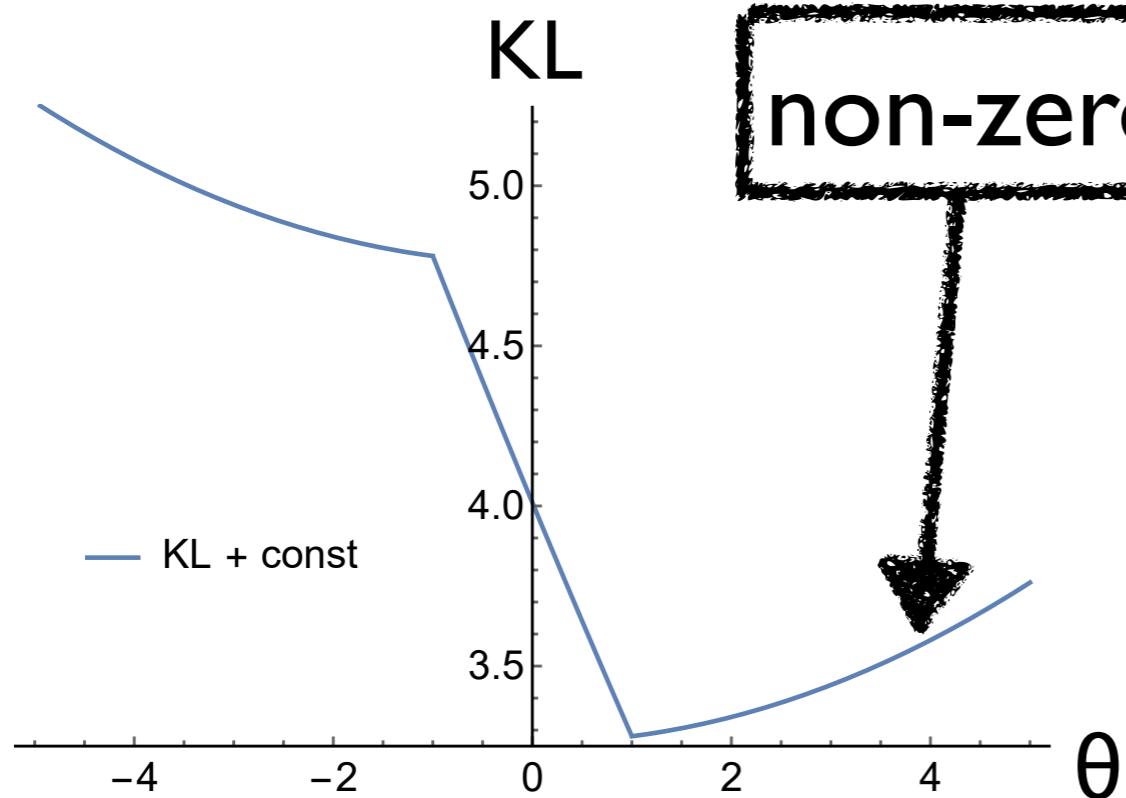
def p(): // model_1
    v = pyro.sample("v", Normal(0., 5.))
    if (v > 0): pyro.sample("obs", Normal(1., 1.), obs=0.)
    else: pyro.sample("obs", Normal(-2., 1.), obs=0.)

```

```

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

```



$$\begin{aligned}
 \nabla_{\theta} \text{KL}[q_{\theta}(z) || p(z|x)]_{\theta=4} \\
 &= (\nabla_{\theta} \log q_{\theta}(z_0))_{\theta=4} \dots \\
 &= (\nabla_{\theta} \log 0.5) \dots \\
 &= 0
 \end{aligned}$$

**How to check that these
bad cases don't happen?**

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

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

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

3. Wrong gradient estimate.

- Biased score estimator due to $\nabla_\theta \int \dots \neq \int \nabla_\theta \dots$

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

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

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

3. Wrong gradient estimate.

- Biased score estimator due to $\nabla_\theta \int \dots \neq \int \nabla_\theta \dots$

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

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

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

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

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

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

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

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

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

3. Wrong gradient estimate.

- Biased score estimator due to $\nabla_\theta \int \dots \neq \int \nabla_\theta \dots$

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 $p(z,x)$.

I. 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 $p(z, x)$.

✓ I. 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 $p(z, x)$.

- ✓ 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 q_theta(): // guide_br'
    theta = pyro.param("theta", 2.)
    sigma = pyro.sample("sigma", Normal(theta, 0.05))

```

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

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

V I. 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 q_theta(): // guide_br'  
    theta = pyro.param("theta", 2.)  
    sigma = pyro.sample("sigma", Normal(theta, 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. $|\mu'(z)| \leq \exp(f(|z|))$ 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 q_theta(): // guide_br'  
    theta = pyro.param("theta", 2.)  
    sigma = pyro.sample("sigma", Normal(theta, 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 .

no 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 .

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

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 $\nabla_{\theta} \int \dots \neq \int \nabla_{\theta} \dots$

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 $\nabla_{\theta} \int \dots \neq \int \nabla_{\theta} \dots$

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 $\nabla_{\theta} \int \dots \neq \int \nabla_{\theta} \dots$

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 .

I. KL not integrable.

2. KL not differentiable

3. Biased due to

$$\int dz \left(\mathcal{N}(z; \dots) \cdot \exp(f(|z|)) \right) < \infty$$

Assume $q_\theta(z)$, $p(z, x)$ for all affine f

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

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

I. 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 .

Sufficient condition

Ensures that KL and score estimator are well-defined, and the estimator is unbiased.

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:
 $p(z|x)=0 \text{ & } q_\theta(z) \neq 0$ 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			θ
			for	plate	sample	score	sample	score	
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 $q\theta$

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 $q\theta$

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 $q\theta$

Reference

The details can be found in our archive paper:

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