

Nordic probabilistic AI school  
Variational Inference and Optimization

Helge Langseth, Andrés Masegosa, and Thomas Dyhre Nielsen

June 18, 2024

# Stochastic Gradient Ascent

## Why do we talk about this?

We want a way to optimize ELBO using gradient methods. If we can do Bayesian inference as optimization it will play well with, e.g., deep learning frameworks.

## Gradient ascent algorithm for maximizing a function $f(\lambda)$ :

- 1 Initialize  $\lambda^{(0)}$  randomly.
- 2 For  $t = 1, \dots$ :

$$\lambda^{(t)} \leftarrow \lambda^{(t-1)} + \rho \cdot \nabla_{\lambda} f(\lambda^{(t-1)})$$

$\lambda^{(t)}$  converges to a (local) optimum of  $f(\cdot)$  if:

- $f$  is “sufficiently nice”;
- The learning-rate  $\rho$  is “sufficiently small”.

### “Standard” gradient ascent is not enough for ELBO optimization

We won't be able to calculate  $\nabla_{\lambda} \mathcal{L}(q(\theta | \lambda))$  exactly for (at least) two reasons:

- 1 We may have to resort to mini-batching (gradient from “random subset”)
- 2 We may not be able to calculate the gradient exactly even for a mini-batch

## “Standard” gradient ascent is not enough for ELBO optimization

We won't be able to calculate  $\nabla_{\lambda} \mathcal{L}(q(\theta | \lambda))$  exactly for (at least) two reasons:

- 1 We may have to resort to mini-batching (gradient from “random subset”)
- 2 We may not be able to calculate the gradient exactly even for a mini-batch

## Stochastic gradient ascent algorithm for maximizing a function $f(\lambda)$ :

If we have access to  $g(\lambda)$  – an **unbiased estimate** of the gradient – it still works!

- 1 Initialize  $\lambda^{(0)}$  randomly.
- 2 For  $t = 1, \dots$ :

$$\lambda^{(t)} \leftarrow \lambda^{(t-1)} + \rho_t \cdot g\left(\lambda^{(t-1)}\right)$$

$\lambda_t$  converges to a (local) optimum of  $f(\cdot)$  if:

- $f$  is “sufficiently nice”;
- $g(\lambda)$  is a random variable with  $\mathbb{E}[g(\lambda)] = \nabla_{\lambda} f(\lambda)$  and  $\text{Var}[g(\lambda)] < \infty$ .
- The learning-rates  $\{\rho_t\}$  is a Robbins-Monro – sequence:
  - $\sum_t \rho_t^2 < \infty$
  - $\sum_t \rho_t = \infty$

# Black Box Variational Inference

## Main idea: Cast inference as an optimization problem

Optimize the ELBO by stochastic gradient ascent over the parameters  $\lambda$ . If that works, Bayesian inference can be **seamlessly integrated** with building-blocks from other gradient-based machine learning approaches (like deep learning).

Algorithm: Maximize  $\mathcal{L}(q) = \mathbb{E}_q \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta|\lambda)} \right]$  by gradient ascent

- Initialization:
  - $t \leftarrow 0$ ;
  - $\hat{\lambda}_0 \leftarrow$  random initialization;
  - $\{\rho_t\} \leftarrow$  a Robbins-Monro sequence.
- Repeat until negligible improvement in terms of  $\mathcal{L}(q)$ :
  - $t \leftarrow t + 1$ ;
  - $\hat{\lambda}_t \leftarrow \hat{\lambda}_{t-1} + \rho_t \nabla_{\lambda} \mathcal{L}(q)|_{\hat{\lambda}_{t-1}}$ ;

## Important issue:

Can we calculate  $\nabla_{\lambda} \mathcal{L}(q)$  efficiently without adding new restrictive assumptions?

The algorithm requires that we can find

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_{\theta \sim q_{\lambda}} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right].$$

**Tricky:** How can we move the gradient inside the expectation?

- We would typically approximate an expectation by a sample average:

$$\mathbb{E}_{\theta \sim q_{\lambda}} [f(\theta, \lambda)] \approx \frac{1}{M} \sum_{j=1}^M f(\theta_j, \lambda), \text{ with } \{\theta_1, \dots, \theta_M\} \text{ sampled from } q_{\lambda}(\theta | \lambda).$$

- This doesn't work when taking a gradient related to the sampling distribution.



The algorithm requires that we can find

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_{\theta \sim q_{\lambda}} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right].$$

**Solution:** Use these properties to simplify the equation:

- ①  $\nabla_{\lambda} (f(\theta, \lambda) \cdot g(\theta, \lambda)) = f(\theta, \lambda) \cdot \nabla_{\lambda} g(\theta, \lambda) + g(\theta, \lambda) \cdot \nabla_{\lambda} f(\theta, \lambda).$
- ②  $\nabla_{\lambda} f(\theta, \lambda) = f(\theta, \lambda) \cdot \nabla_{\lambda} \log f(\theta, \lambda).$
- ③  $\mathbb{E}_q [\nabla_{\lambda} \log q(\theta | \lambda)] = 0$  for any density function  $q(\theta | \lambda).$

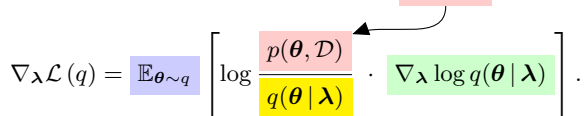
Now it follows that

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{\theta \sim q_{\lambda}} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \cdot \nabla_{\lambda} \log q(\theta | \lambda) \right].$$

This is the so-called **score-function gradient**.

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{\boldsymbol{\theta} \sim q} \left[ \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} | \boldsymbol{\lambda})} \cdot \nabla_{\lambda} \log q(\boldsymbol{\theta} | \boldsymbol{\lambda}) \right].$$

- We still only need access to the joint distribution  $p(\boldsymbol{\theta}, \mathcal{D})$  – not  $p(\boldsymbol{\theta} | \mathcal{D})$ .

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) = \mathbb{E}_{\boldsymbol{\theta} \sim q} \left[ \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} | \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} | \boldsymbol{\lambda}) \right].$$


- We still only need access to the joint distribution  $p(\boldsymbol{\theta}, \mathcal{D})$  – not  $p(\boldsymbol{\theta} | \mathcal{D})$ .

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) = \mathbb{E}_{\boldsymbol{\theta} \sim q} \left[ \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} | \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} | \boldsymbol{\lambda}) \right].$$

- $q(\boldsymbol{\theta} | \boldsymbol{\lambda})$  factorizes under MF, s.t. we can optimize per variable:  $q(\theta_i | \boldsymbol{\lambda}_i)$ .

- We still only need access to the joint distribution  $p(\boldsymbol{\theta}, \mathcal{D})$  – not  $p(\boldsymbol{\theta} | \mathcal{D})$ .

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) = \mathbb{E}_{\boldsymbol{\theta} \sim q} \left[ \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} | \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} | \boldsymbol{\lambda}) \right].$$

- $q(\boldsymbol{\theta} | \boldsymbol{\lambda})$  factorizes under MF, s.t. we can optimize per variable:  $q(\theta_i | \boldsymbol{\lambda}_i)$ .
- We must calculate  $\nabla_{\boldsymbol{\lambda}_i} \log q(\theta_i | \boldsymbol{\lambda}_i)$ , which is also known as the “score function”.

- We still only need access to the joint distribution  $p(\boldsymbol{\theta}, \mathcal{D})$  – not  $p(\boldsymbol{\theta} | \mathcal{D})$ .

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) = \mathbb{E}_{\boldsymbol{\theta} \sim q} \left[ \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} | \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} | \boldsymbol{\lambda}) \right].$$

- $q(\boldsymbol{\theta} | \boldsymbol{\lambda})$  factorizes under **MF**, s.t. we can optimize per variable:  $q(\theta_i | \boldsymbol{\lambda}_i)$ .
- We must calculate  $\nabla_{\boldsymbol{\lambda}_i} \log q(\theta_i | \boldsymbol{\lambda}_i)$ , which is also known as the “score function”.
- The expectation will be approximated using a sample  $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M\}$  generated from  $q(\boldsymbol{\theta} | \boldsymbol{\lambda})$ . Hence we require that we can **sample from** each  $q(\theta_i | \boldsymbol{\lambda}_i)$ .

- We still only need access to the joint distribution  $p(\boldsymbol{\theta}, \mathcal{D})$  – not  $p(\boldsymbol{\theta} | \mathcal{D})$ .

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) = \mathbb{E}_{\boldsymbol{\theta} \sim q} \left[ \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} | \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} | \boldsymbol{\lambda}) \right].$$

- $q(\boldsymbol{\theta} | \boldsymbol{\lambda})$  factorizes under **MF**, s.t. we can optimize per variable:  $q(\theta_i | \boldsymbol{\lambda}_i)$ .
- We must calculate  $\nabla_{\boldsymbol{\lambda}_i} \log q(\theta_i | \boldsymbol{\lambda}_i)$ , which is also known as the “score function”.
- The expectation will be approximated using a sample  $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M\}$  generated from  $q(\boldsymbol{\theta} | \boldsymbol{\lambda})$ . Hence we require that we can **sample from** each  $q(\theta_i | \boldsymbol{\lambda}_i)$ .

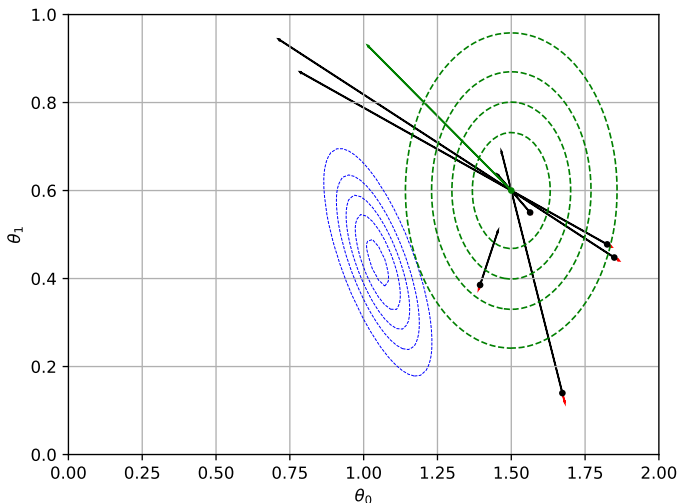
## Calculating the gradient – in summary

We have observed the data  $\mathcal{D}$ , and our current estimate for  $\boldsymbol{\lambda}$  is  $\hat{\boldsymbol{\lambda}}$ . Then

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q)|_{\boldsymbol{\lambda}=\hat{\boldsymbol{\lambda}}} \approx \frac{1}{M} \sum_{j=1}^M \log \frac{p(\boldsymbol{\theta}_j, \mathcal{D})}{q(\boldsymbol{\theta}_j | \hat{\boldsymbol{\lambda}})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_j | \hat{\boldsymbol{\lambda}}),$$

where  $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M\}$  are samples from  $q(\cdot | \hat{\boldsymbol{\lambda}})$ . Typically  $M$  is fairly small.

# Does it work?

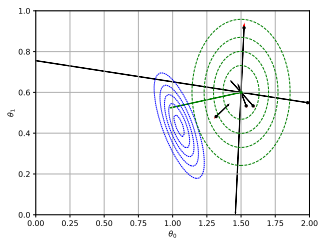
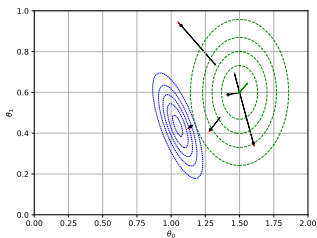
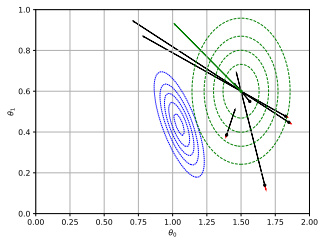
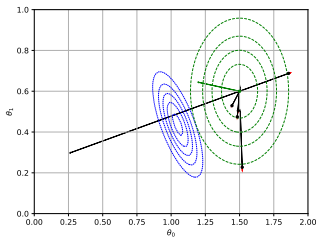


$$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i | \boldsymbol{\lambda}); \quad \log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i | \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i | \boldsymbol{\lambda}); \quad \frac{1}{M} \sum_{j=1}^M \log \frac{p(\boldsymbol{\theta}_j, \mathcal{D})}{q(\boldsymbol{\theta}_j | \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_j | \boldsymbol{\lambda})$$

Length of gradients increased for visibility. Graphics inspired by Arto Klami @ ProbAI2021.



# Does it work?

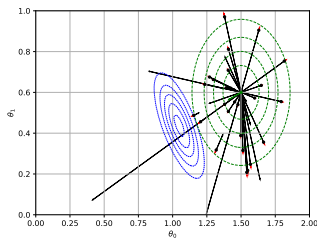
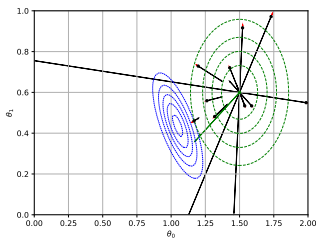
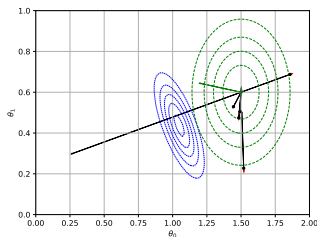
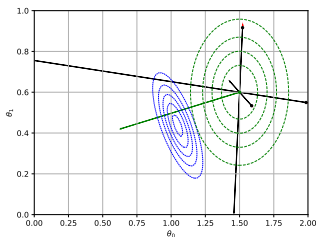


Different samples, each with  $M = 5$ .

$$\nabla_{\lambda} \log q(\theta_i | \lambda); \log \frac{p(\theta_i, \mathcal{D})}{q(\theta_i | \lambda)} \cdot \nabla_{\lambda} \log q(\theta_i | \lambda); \frac{1}{M} \sum_{j=1}^M \log \frac{p(\theta_j, \mathcal{D})}{q(\theta_j | \lambda)} \cdot \nabla_{\lambda} \log q(\theta_j | \lambda)$$

Length of gradients increased for visibility. Graphics inspired by Arto Klami @ ProbAI2021.

# Does it work?



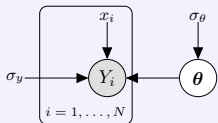
Different values of  $M$  ( $M = 3, 5, 10$ , and  $25$ )

$$\nabla_{\lambda} \log q(\theta_i | \lambda); \log \frac{p(\theta_i, \mathcal{D})}{q(\theta_i | \lambda)} \cdot \nabla_{\lambda} \log q(\theta_i | \lambda); \frac{1}{M} \sum_{j=1}^M \log \frac{p(\theta_j, \mathcal{D})}{q(\theta_j | \lambda)} \cdot \nabla_{\lambda} \log q(\theta_j | \lambda)$$

Length of gradients increased for visibility. Graphics inspired by Arto Klami @ ProbAI2021.

# Does it work?

## Code Task: Score-function gradient for linear regression



- $\boldsymbol{\theta} = \{w_0, w_1\}$ ,  $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \sigma_\theta \cdot \mathbf{I}_{2 \times 2})$
- $Y_i \mid \{\boldsymbol{\theta}, x_i, \sigma_y\} \sim \mathcal{N}(w_0 + w_1 \cdot x_i, \sigma_y^2)$
- We choose  $q_j(\theta_j \mid \boldsymbol{\lambda}_j) = \mathcal{N}(\theta_j \mid \mu_j, \sigma_j^2)$ , so  $\boldsymbol{\lambda}_j = \{\mu_j, \sigma_j\}$

In this task you will implement the score-function gradient:

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) = \mathbb{E}_{\boldsymbol{\theta} \sim q} \left[ \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \right].$$

- Look at `Exercise 1` in the notebook

`Day2-AfterLunch/students_BBVI.ipynb`.

- We need  $\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) = \left[ \frac{\partial}{\partial \mu} \log \mathcal{N}(\theta \mid \mu, \sigma^2), \frac{\partial}{\partial \sigma} \log \mathcal{N}(\theta \mid \mu, \sigma^2) \right]^\top$ :  
You must find  $\frac{\partial}{\partial \mu_j} \log \mathcal{N}(\theta_j \mid \mu_j, \sigma_j^2)$ , but are given  $\frac{\partial}{\partial \sigma_j} \log \mathcal{N}(\theta_j \mid \mu_j, \sigma_j^2)$ .
- Implement your results in the function `score_function_gradient`.

Goal: Find a more robust estimator for the gradient

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_{\theta \sim q} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right].$$

Goal: Find a more robust estimator for the gradient

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_{\theta \sim q} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right].$$

**Assumption:**  $q(\theta | \lambda)$  can be *reparametrized* as follows:

$$\begin{aligned}\epsilon &\sim \phi(\epsilon) \\ \theta &= f(\epsilon, \lambda),\end{aligned}$$

where  $\phi(\epsilon)$  is some distribution that **does not** depend on  $\lambda$  and  $f(\epsilon, \lambda)$  is a **deterministic** transformation.

Goal: Find a more robust estimator for the gradient

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_{\theta \sim q} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right].$$

**Assumption:**  $q(\theta | \lambda)$  can be *reparametrized* as follows:

$$\begin{aligned}\epsilon &\sim \phi(\epsilon) \\ \theta &= f(\epsilon, \lambda),\end{aligned}$$

where  $\phi(\epsilon)$  is some distribution that **does not** depend on  $\lambda$  and  $f(\epsilon, \lambda)$  is a **deterministic** transformation.

Example: The univariate Gaussian distribution

Assume  $q(\theta | \lambda) = \mathcal{N}(\theta | \mu, \sigma^2)$ , with  $\lambda = \{\mu, \sigma\}$ .  $q(\theta | \lambda)$  can be reparametrized by

$$\begin{aligned}\epsilon \sim \phi(\epsilon) &= \mathcal{N}(0, 1) \\ \theta = f(\epsilon, \lambda) &= \mu + \sigma \epsilon.\end{aligned}$$

**Assumption:**  $q(\theta|\lambda)$  can be *reparametrized* as follows:

$$\begin{aligned}\epsilon &\sim \phi(\epsilon) \\ \theta &= f(\epsilon, \lambda).\end{aligned}$$

**Now we can calculate the gradient:**

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_{\theta \sim q(\cdot | \lambda)} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right]$$



**Assumption:**  $q(\theta|\lambda)$  can be *reparametrized* as follows:

$$\begin{aligned}\epsilon &\sim \phi(\epsilon) \\ \theta &= f(\epsilon, \lambda).\end{aligned}$$

**Now we can calculate the gradient:**

$$\begin{aligned}\nabla_{\lambda} \mathcal{L}(q) &= \nabla_{\lambda} \mathbb{E}_{\theta \sim q(\cdot | \lambda)} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right] \\ &= \nabla_{\lambda} \mathbb{E}_{\epsilon \sim \phi(\cdot)} \left[ \log \frac{p(f(\epsilon, \lambda), \mathcal{D})}{q(f(\epsilon, \lambda) | \lambda)} \right]\end{aligned}$$

**Assumption:**  $q(\theta|\lambda)$  can be *reparametrized* as follows:

$$\begin{aligned}\epsilon &\sim \phi(\epsilon) \\ \theta &= f(\epsilon, \lambda).\end{aligned}$$

**Now we can calculate the gradient:**

$$\begin{aligned}\nabla_{\lambda} \mathcal{L}(q) &= \nabla_{\lambda} \mathbb{E}_{\theta \sim q(\cdot | \lambda)} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right] \\ &= \mathbb{E}_{\epsilon \sim \phi(\cdot)} \left[ \nabla_{\lambda} \log \frac{p(f(\epsilon, \lambda), \mathcal{D})}{q(f(\epsilon, \lambda) | \lambda)} \right] \\ &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\lambda} \log \frac{p(f(\epsilon, \lambda), \mathcal{D})}{q(f(\epsilon, \lambda) | \lambda)} \right]\end{aligned}$$

**Assumption:**  $q(\theta|\lambda)$  can be *reparametrized* as follows:

$$\begin{aligned}\epsilon &\sim \phi(\epsilon) \\ \theta &= f(\epsilon, \lambda).\end{aligned}$$

**Now we can calculate the gradient:**

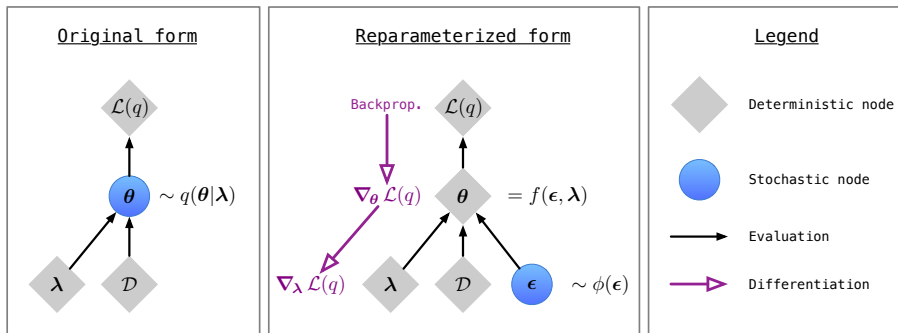
$$\begin{aligned}\nabla_{\lambda} \mathcal{L}(q) &= \nabla_{\lambda} \mathbb{E}_{\theta \sim q(\cdot | \lambda)} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \right] \\ &= \nabla_{\lambda} \mathbb{E}_{\epsilon \sim \phi(\cdot)} \left[ \log \frac{p(f(\epsilon, \lambda), \mathcal{D})}{q(f(\epsilon, \lambda) | \lambda)} \right] \\ &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\lambda} \log \frac{p(f(\epsilon, \lambda), \mathcal{D})}{q(f(\epsilon, \lambda) | \lambda)} \right] \\ &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \Big|_{\theta=f(\epsilon, \lambda)} \cdot \nabla_{\lambda} f(\epsilon, \lambda) \right]\end{aligned}$$

# The Reparametrization Trick

**Assumption:**  $q(\theta|\lambda)$  can be *reparametrized* as follows:

$$\begin{aligned}\epsilon &\sim \phi(\epsilon) \\ \theta &= f(\epsilon, \lambda).\end{aligned}$$

Now we can calculate the gradient:



## Monte-Carlo Estimation:

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \nabla_{\lambda} f(\epsilon, \lambda) \right]$$

## Monte-Carlo Estimation:

$$\begin{aligned}\nabla_{\lambda} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \nabla_{\lambda} f(\epsilon, \lambda) \right] \\ &\approx \frac{1}{M} \sum_{j=1}^M \left[ \nabla_{\theta} \log \frac{p(\theta_j, \mathcal{D})}{q(\theta_j | \lambda)} \nabla_{\lambda} f(\epsilon_j, \lambda) \right] : \epsilon_j \sim \phi(\epsilon), \theta_j \leftarrow f(\epsilon_j, \lambda)\end{aligned}$$

## Monte-Carlo Estimation:

$$\begin{aligned}\nabla_{\lambda} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \nabla_{\lambda} f(\epsilon, \lambda) \right] \\ &\approx \frac{1}{M} \sum_{j=1}^M \left[ \nabla_{\theta} \log \frac{p(\theta_j, \mathcal{D})}{q(\theta_j | \lambda)} \nabla_{\lambda} f(\epsilon_j, \lambda) \right] : \epsilon_j \sim \phi(\epsilon), \quad \theta_j \leftarrow f(\epsilon_j, \lambda) \\ &= \frac{1}{M} \sum_{j=1}^M \left( \underbrace{\nabla_{\theta} \log p(\theta_j, \mathcal{D})}_{\text{Model's gradient}} - \underbrace{\nabla_{\theta} \log q(\theta_j | \lambda)}_{\text{Approximation's gradient}} \right) \cdot \nabla_{\lambda} f(\epsilon_j, \lambda)\end{aligned}$$

## Monte-Carlo Estimation:

$$\begin{aligned}\nabla_{\lambda} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \nabla_{\lambda} f(\epsilon, \lambda) \right] \\ &\approx \frac{1}{M} \sum_{j=1}^M \left[ \nabla_{\theta} \log \frac{p(\theta_j, \mathcal{D})}{q(\theta_j | \lambda)} \nabla_{\lambda} f(\epsilon_j, \lambda) \right] \quad : \quad \epsilon_j \sim \phi(\epsilon), \quad \theta_j \leftarrow f(\epsilon_j, \lambda) \\ &= \frac{1}{M} \sum_{j=1}^M \left( \underbrace{\nabla_{\theta} \log p(\theta_j, \mathcal{D})}_{\text{Model's gradient}} - \underbrace{\nabla_{\theta} \log q(\theta_j | \lambda)}_{\text{Approximation's gradient}} \right) \cdot \nabla_{\lambda} f(\epsilon_j, \lambda)\end{aligned}$$

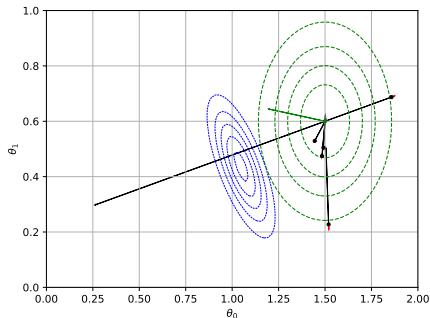
## This gradient estimator...

- Uses the *model's* gradients (not so for the score-function gradient).
- Requires  $q(\theta|\lambda)$  to be *reparametrizable* and *differentiable* – this time wrt.  $\theta$ .
- Requires  $\log p(\theta, \mathcal{D})$  to be differentiable wrt.  $\theta$ .

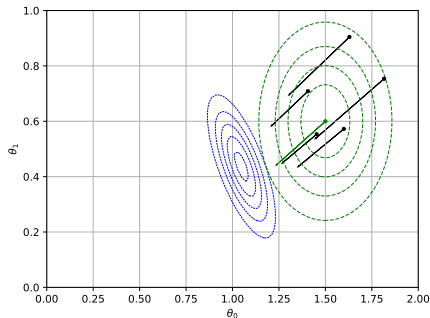


# Does it work?

Score-function gradient



Reparameterized gradient



Length of gradients increased for visibility. Graphics inspired by Arto Klami @ ProbAI2021.

Notice the direction of each sample's gradient:

- **Score-function gradient:** Towards the mode of  $q$
- **Reparameterization-gradient:** (Approximately) towards high density region of the exact posterior  $p(\theta|\mathcal{D})$ .

## Score function gradients:

- Gradients point towards the mode of  $q(\theta|\lambda)$ , while  $p(\mathcal{D}, \theta)$  only affects the *weights*. We need a “large” number of samples, typically in the order of tens to a hundred.
- **Requires**  $\ln q(\theta|\lambda)$  to be *differentiable wrt.  $\lambda$* .
- **Requires**  $\ln p(\mathcal{D}, \theta)$  to be *computable*.

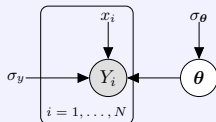
## Reparametrization gradients:

- Gradients utilize the model definition via the term  $\nabla_{\theta} \ln p(\mathcal{D}, \theta)$ . Fairly robust, so we only need a *few samples*, typically only a single one!
- **Requires**  $q(\theta|\lambda)$  to be *reparametrizable*.
- **Requires**  $\ln q(\theta|\lambda)$  to be *differentiable wrt.  $\theta$* .
- **Requires**  $\ln p(\mathcal{D}, \theta)$  to be *differentiable wrt.  $\theta$* .

## Conclusion

The “Score function” approach is more general, but “Reparametrization” will usually provide better results quicker when applicable.

## Code Task: Reparameterization-gradient for linear regression



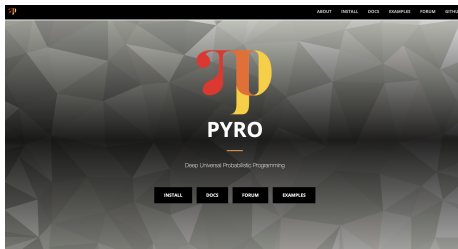
- $\theta = \{w_0, w_1\}$ ,  $\theta \sim \mathcal{N}(\mathbf{0}, \sigma_\theta \cdot \mathbf{I}_{2 \times 2})$
- $Y_i \mid \{\theta, x_i, \sigma_y\} \sim \mathcal{N}(w_0 + w_1 \cdot x_i, \sigma_y^2)$

In this task you will play with the reparameterization gradient:

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{\epsilon \sim \phi} [(\nabla_{\theta} \log p(\theta, \mathcal{D}) - \nabla_{\theta} \log q(\theta \mid \lambda)) \nabla_{\lambda} f(\epsilon, \lambda)]$$

- We provide  $\nabla_{\theta} \log p(\theta, \mathcal{D})$ ,  $\nabla_{\theta} \log q(\theta \mid \lambda)$  and  $\nabla_{\lambda} f(\epsilon, \lambda)$  for this model.
- Go to Exercise 2 in `Day2-AfterLunch/students_BBVI.ipynb`.
- Experiment with the number of Monte-Carlo samples  $M$  per iteration, the learning-rate, and the number of iterations. Compare with the output of the Score Function Gradient.

# Probabilistic programming: Variational inference in Pyro



## Pyro's main features ([www.pyro.ai](http://www.pyro.ai)) :

- Initially developed by UBER (the car riding company).
- Community of contributors and a dedicated team at Broad Institute (US).
- Rely on Pytorch (Deep Learning Framework).
- Enable GPU acceleration and distributed learning.

<https://github.com/PGM-Lab/2024-ProbAI>

## Pyro

Pyro ([pyro.ai](http://pyro.ai)) is a Python library for probabilistic modeling, inference, and criticism, integrated with PyTorch.

- Modeling:**
  - Directed graphical models
  - Neural networks (via `nn.Module`)
  - ...
- Inference:**
  - Variational inference – including BBVI, SVI
  - Monte Carlo – including Importance sampling and Hamiltonian Monte Carlo
  - ...
- Criticism:**
  - Point-based evaluations
  - Posterior predictive checks
  - ...

... and there are also many other possibilities

Tensorflow is integrating probabilistic thinking into its core, InferPy is a local alternative, etc.

## Inference Problem

$$p(\text{temp} | \text{sensor} = 18)$$

## Inference Problem

$$p(\text{temp}|\text{sensor} = 18)$$

## Variational Solution

$$\min_{\underset{q}{q}} \text{KL}(q(\text{temp})||p(\text{temp}|\text{sensor} = 18))$$



## Inference Problem

$$p(\text{temp}|\text{sensor} = 18)$$

## Variational Solution

$$\min_{\underset{q}{q}} \text{KL} (q(\text{temp}) || p(\text{temp}|\text{sensor} = 18))$$

## Pyro Guides:

- Define the  $q$  **distributions** in variational settings.

## Inference Problem

$$p(\text{temp}|\text{sensor} = 18)$$

## Variational Solution

$$\min_q \text{KL} (q(\text{temp}) || p(\text{temp}|\text{sensor} = 18))$$

## Pyro Guides:

- Define the  $q$  **distributions** in variational settings.
- Build **proposal distributions** in importance sampling, MCMC.
- ...

## Pyro Guides:

- Guides are **arbitrary stochastic functions**.
- Guides produces samples for those variables of the model which are **not observed**.

## Pyro Guides:

- Guides are **arbitrary stochastic functions**.
- Guides produces samples for those variables of the model which are **not observed**.

## Guide requirements

- 1 the guide has the same input signature as the model
- 2 all unobserved sample statements that appear in the model appear in the guide.

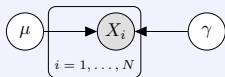
## Example

```
1 #The observations
2 obs = {'sensor': torch.tensor(18.0)}
3
4 def model(obs):
5     temp = pyro.sample('temp', dist.Normal(15.0, 2.0))
6     sensor = pyro.sample('sensor', dist.Normal(temp, 1.0), obs=obs['sensor'])
```

```
1 #The guide
2 def guide(obs):
3     a = pyro.param("mean", torch.tensor(0.0))
4     b = pyro.param("scale", torch.tensor(1.), constraint=constraints.positive)
5     temp = pyro.sample('temp', dist.Normal(a, b))
```

**Exercise: Pyro implementation for a simple Gaussian model**

Day2-AfterLunch/student\_simple\_gaussian\_model\_pyro.ipynb



- $X_i \mid \{\mu, \gamma\} \sim \mathcal{N}(\mu, 1/\gamma)$
- $\mu \sim \mathcal{N}(0, \tau)$
- $\gamma \sim \text{Gamma}(\alpha, \beta)$

- Implement a pyro **guide** for the graphical model above.
- Specify suitable **variational approximation** in the form of a Pyro guide.

$$q(\mu, \gamma) = \dots$$

- **Check** the differences with the following notebook (no Pyro implementation).

Day2-BeforeLunch/student\_simple\_model.ipynb