Probabilistic Machine learning

Variational Inference and Learning

Andrés Masegosa and Thomas Dyhre Nielsen

September 22, 2023

Stochastic Gradient Ascent

A small side-step: 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 $f(\boldsymbol{\lambda})$:

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

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

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

- f is "sufficiently nice";
- The learning-rate ρ is "sufficiently small".

. . . and Stochastic Gradient Ascent

"Standard" gradient ascent is not enough for ELBO optimization

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

1. We may have to resolve 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 $\nabla_{\boldsymbol{\lambda}} \mathcal{L}\left(q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})\right)$ exactly for (at least) two reasons:

1. We may have to resolve 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 $f(\boldsymbol{\lambda})$:

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

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

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

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

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

Black Box Variational Inference

Main idea: Cast inference as an optimization problem

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

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

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

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

BBVI - calculating the gradient

The algorithm requires that we can find

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

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

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

$$\mathbb{E}_{\boldsymbol{\theta} \sim q_{\boldsymbol{\lambda}}} [f(\boldsymbol{\theta}, \boldsymbol{\lambda})] \approx \frac{1}{M} \sum_{j=1}^{M} f(\boldsymbol{\theta}_j, \boldsymbol{\lambda}), \text{ with } \{\boldsymbol{\theta}_1, \ldots \boldsymbol{\theta}_M\} \text{ sampled from } q_{\boldsymbol{\lambda}}(\boldsymbol{\theta} \mid \boldsymbol{\lambda}).$$

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

BBVI - calculating the gradient

The algorithm requires that we can find

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

Solution: Use these properties to simplify the equation:

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

Now it follows that

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

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

Calculating the gradient – Things to notice

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

Calculating the gradient – Things to notice

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

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

Calculating the gradient – Things to notice

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

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

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

Calculating the gradient – Things to notice

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

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

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

Calculating the gradient – Things to notice

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

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

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

Calculating the gradient – Things to notice

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

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

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

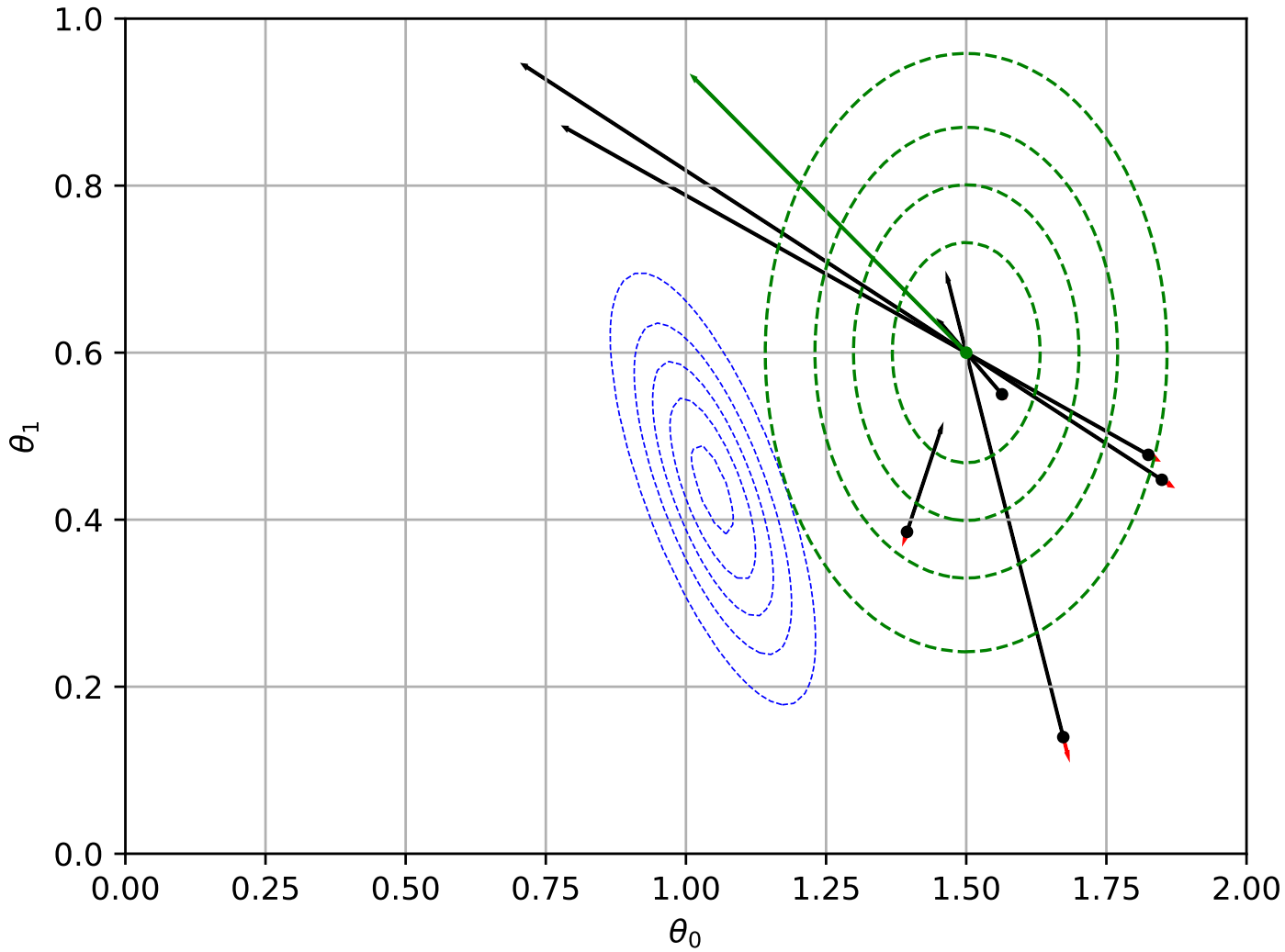
Calculating the gradient – in summary

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

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

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

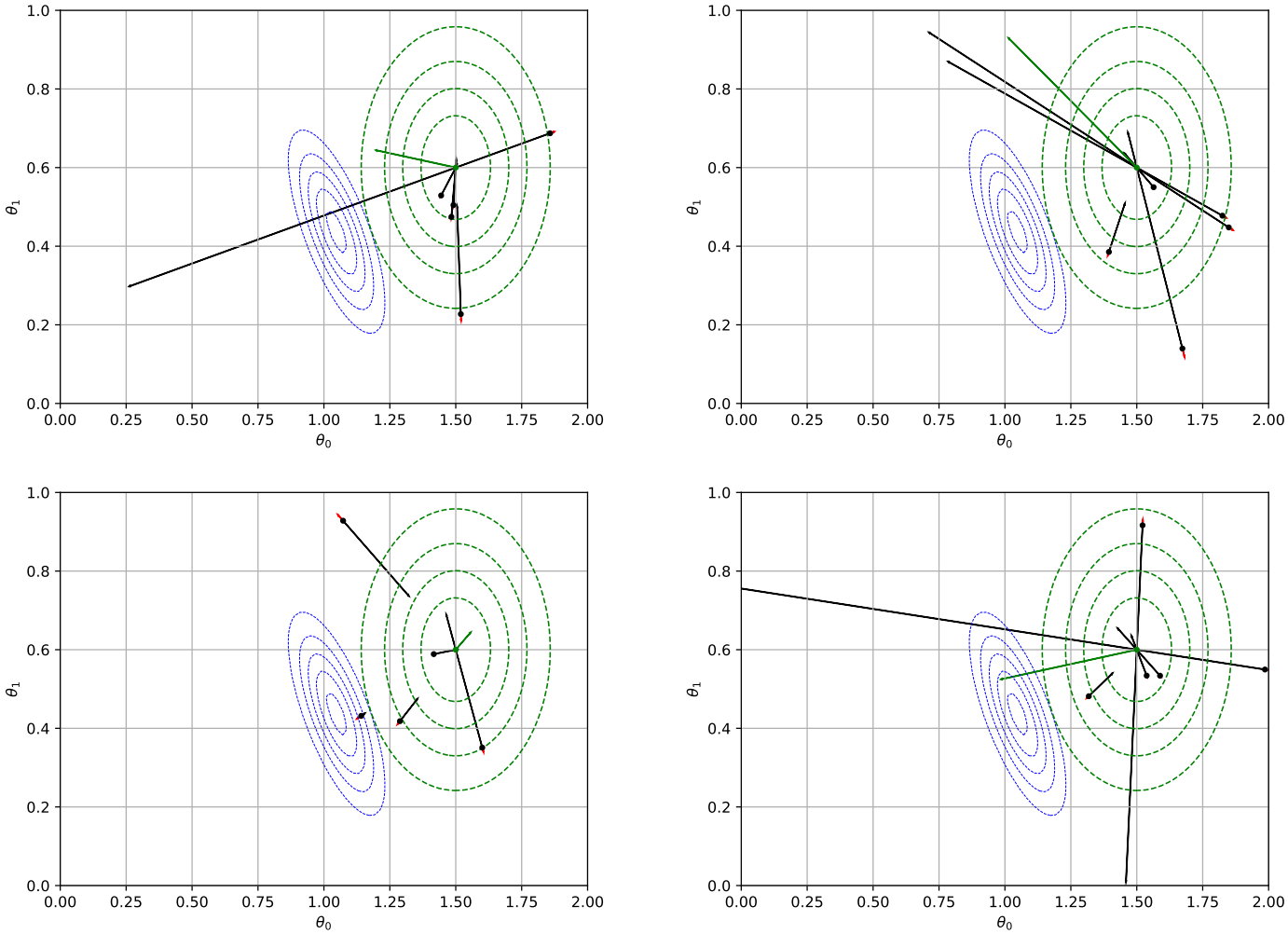
Does it work?



$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\frac{1}{M} \sum_{i=1}^{m} \log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$

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

Does it work?

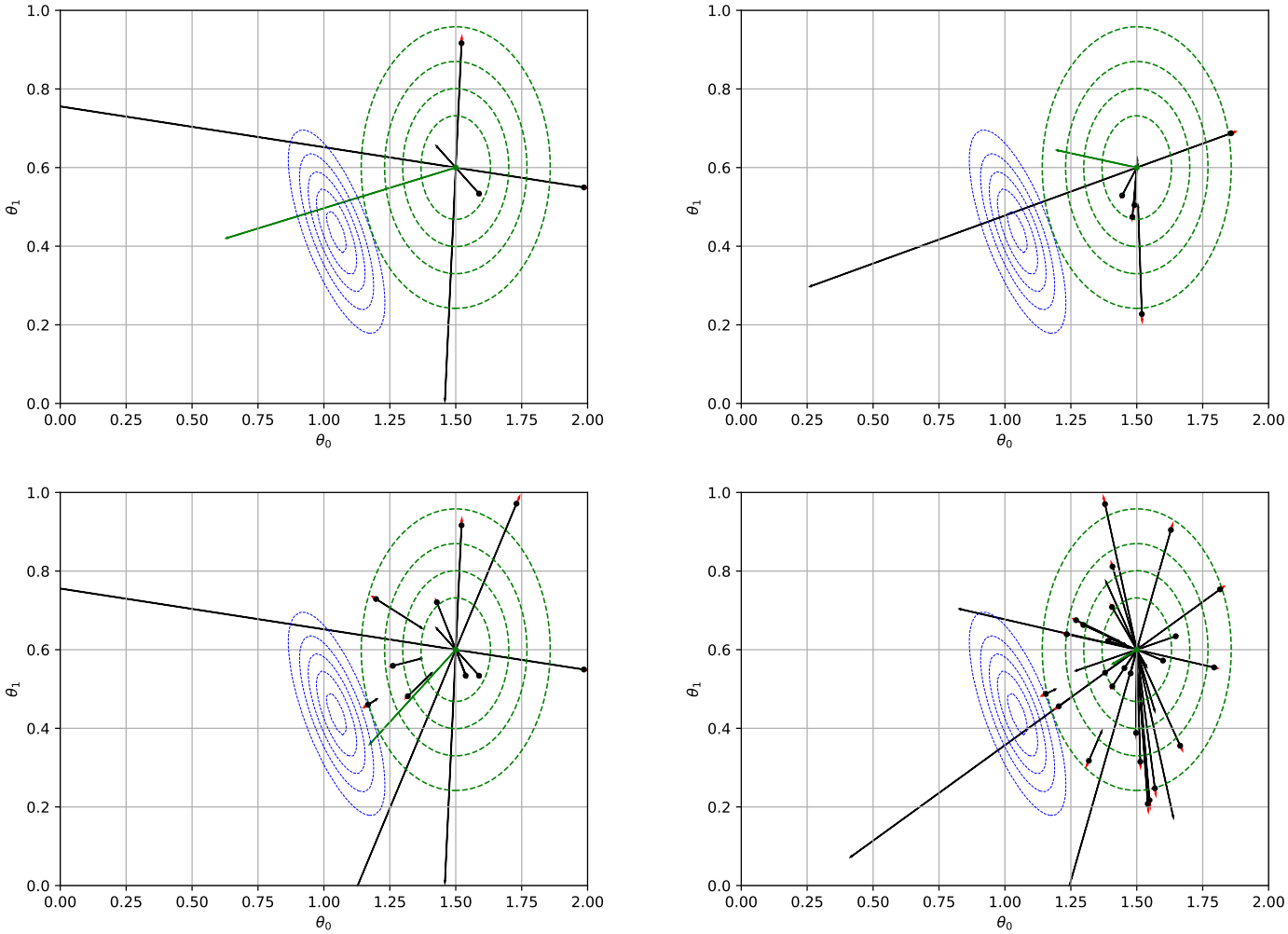


Different samples, each with $M = 5$.

$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\frac{1}{M} \sum_{i=1}^{m} \log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$

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

Does it work?

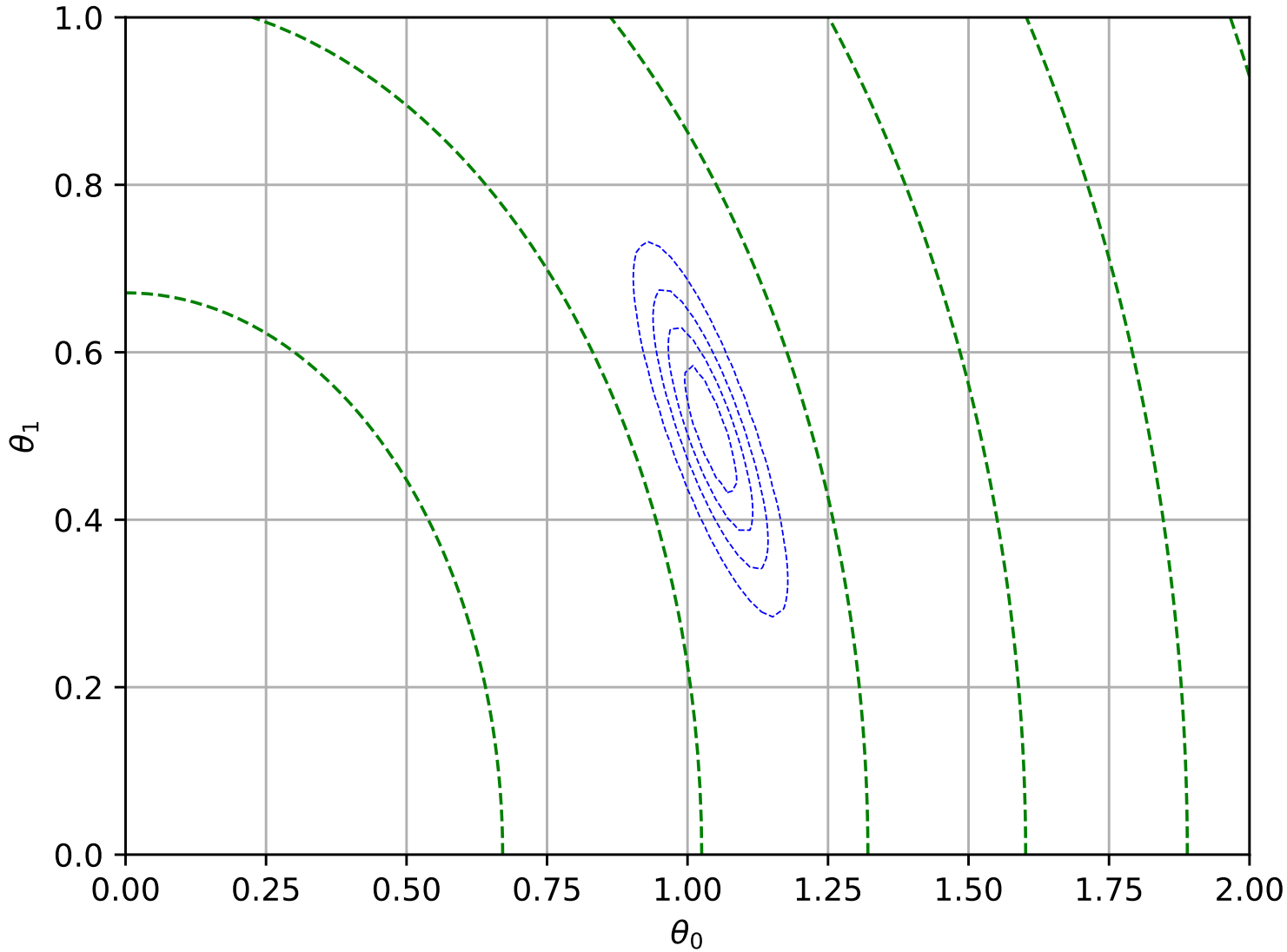


Different values of M ($M = 3$, 5, 10, and 25)

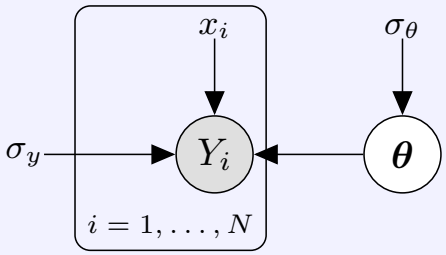
$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\frac{1}{M} \sum_{i=1}^{m} \log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$

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

Does it work?


1.0
0.8
0.6
0.4
0.2
0.0
θ_1
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
θ_0

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`.
- Calculate $\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$, i.e., $\frac{\partial}{\partial \mu} \log \mathcal{N}(\mu, \sigma^2)$ and $\frac{\partial}{\partial \sigma} \log \mathcal{N}(\mu, \sigma^2)$ by hand.
- Implement your results in the function `score_function_gradient`.

The Reparametrization Trick

Let's try to find another trick to compute:

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

The Reparametrization Trick

Let's try to find another trick to compute:

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

Let's assume $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

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

The Reparametrization Trick

Let's try to find another trick to compute:

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

Let's assume $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

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

The common example is $q(\boldsymbol{\theta}|\boldsymbol{\lambda}) = \mathcal{N}(\mu, \sigma)$ *reparametrized* using

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

The Reparametrization Trick

If $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

The Reparametrization Trick

If $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

Now we can do something different:

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

The Reparametrization Trick

If $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

Now we can do something different:

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

The Reparametrization Trick

If $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

Now we can do something different:

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

The Reparametrization Trick

If $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

Now we can do something different:

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

(slide 5 - point 3)

The Reparametrization Trick

If $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be *reparametrized*:

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

Now we can do something different:

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{\boldsymbol{\theta}\sim q}\left[\log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})}\right] \\ &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{\epsilon\sim \phi}\left[\log \frac{p(f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}), \mathcal{D})}{q(f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}) \mid \boldsymbol{\lambda})}\right] \\ &= \mathbb{E}_{\epsilon\sim \phi}\left[\nabla_{\boldsymbol{\lambda}} \log \frac{p(f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}), \mathcal{D})}{q(f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}) \mid \boldsymbol{\lambda})}\right] \\ &= \mathbb{E}_{\epsilon\sim \phi}\left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}) + \nabla_{\boldsymbol{\lambda}} \log q(f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})|\boldsymbol{\lambda})\right] \quad \text{(slide 5 - point 3)} \\ &= \mathbb{E}_{\epsilon\sim \phi}\left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\right] \end{aligned}$$

The Reparametrization Trick

Monte-Carlo Estimation:

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

Monte-Carlo Estimation:

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

Monte-Carlo Estimation:

$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi}\left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\right] \\
&\approx \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}_j, \mathcal{D})}{q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \quad : \boldsymbol{\epsilon}_j \sim \phi(\boldsymbol{\epsilon}),\ \boldsymbol{\theta}_j = f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \\
&= \frac{1}{M} \sum_{j=1}^{M} \Big(\underbrace{\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_j, \mathcal{D})}_{\text{Model's Gradient}} - \nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda}) \Big) \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda})
\end{aligned}
$$

Monte-Carlo Estimation:

$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi}\left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\right] \\
&\approx \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}_j, \mathcal{D})}{q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \quad : \boldsymbol{\epsilon}_j \sim \phi(\boldsymbol{\epsilon}),\ \boldsymbol{\theta}_j = f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \\
&= \frac{1}{M} \sum_{j=1}^{M} \Big(\underbrace{\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_j, \mathcal{D})}_{\text{Model's Gradient}} - \nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda}) \Big) \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda})
\end{aligned}
$$

This gradient estimator directly uses **model's gradients**

Monte-Carlo Estimation:

$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi}\left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\right] \\
&\approx \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}_j, \mathcal{D})}{q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \quad : \boldsymbol{\epsilon}_j \sim \phi(\boldsymbol{\epsilon}),\ \boldsymbol{\theta}_j = f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \\
&= \frac{1}{M} \sum_{j=1}^{M} \Big(\underbrace{\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_j, \mathcal{D})}_{\text{Model's Gradient}} - \nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda}) \Big) \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda})
\end{aligned}
$$

This gradient estimator directly uses **model's gradients**

- While the **score function estimator** does not.

Monte-Carlo Estimation:

$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi}\left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\right] \\
&\approx \frac{1}{M} \sum_{j=1}^{M} \nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}_j, \mathcal{D})}{q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \quad : \boldsymbol{\epsilon}_j \sim \phi(\boldsymbol{\epsilon}),\ \boldsymbol{\theta}_j = f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda}) \\
&= \frac{1}{M} \sum_{j=1}^{M} \Big(\underbrace{\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_j, \mathcal{D})}_{\text{Model's Gradient}} - \nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{\theta}_j \mid \boldsymbol{\lambda}) \Big) \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}_j, \boldsymbol{\lambda})
\end{aligned}
$$

This gradient estimator directly uses **model's gradients**

- While the **score function estimator** does not.
- $\log p(\boldsymbol{\theta}, \mathcal{D})$ needs to be differentiable wrt $\boldsymbol{\theta}$ (i.e. **no discrete variables**).
- $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ needs to be **differentiable** and **reparametrizable**

Reparameterization can be done for a **(growing) set of distributions**:

Target	$p(z;\theta)$	Base $p(\epsilon)$	One-liner $g(\epsilon;\theta)$
Exponential	$\exp(-x); x > 0$	$\epsilon \sim [0;1]$	$\ln(1/\epsilon)$
Cauchy	$\frac{1}{\pi(1+x^2)}$	$\epsilon \sim [0;1]$	$\tan(\pi\epsilon)$
Laplace	$\mathcal{L}(0;1) = \exp(-\lvert x\rvert)$	$\epsilon \sim [0;1]$	$\ln(\frac{\epsilon_1}{\epsilon_2})$
Laplace	$\mathcal{L}(\mu;b)$	$\epsilon \sim [0;1]$	$\mu - bsgn(\epsilon)\ln(1-2\lvert\epsilon\rvert)$
Std Gaussian	$\mathcal{N}(0;1)$	$\epsilon \sim [0;1]$	$\sqrt{\ln(\frac{1}{\epsilon_1})}\cos(2\pi\epsilon_2)$
Gaussian	$\mathcal{N}(\mu;RR^\top)$	$\epsilon \sim \mathcal{N}(0;1)$	$\mu + R\epsilon$
Rademacher	$Rad(\frac{1}{2})$	$\epsilon \sim Bern(\frac{1}{2})$	$2\epsilon - 1$
Log-Normal	$\ln \mathcal{N}(\mu;\sigma)$	$\epsilon \sim \mathcal{N}(\mu;\sigma^2)$	$\exp(\epsilon)$
Inv Gamma	$i\mathcal{G}(k;\theta)$	$\epsilon \sim \mathcal{G}(k;\theta^{-1})$	$\frac{1}{\epsilon}$

Table from http://blog.shakirm.com/2015/10/ machine-learning-trick-of-the-day-4-reparameterisation-tricks/

Reparametrizable Distributions

Reparameterization can be done for a **(growing) set of distributions**:

Target	$p(z;\theta)$	Base $p(\epsilon)$	One-liner $g(\epsilon;\theta)$
Exponential	$\exp(-x); x > 0$	$\epsilon \sim [0;1]$	$\ln(1/\epsilon)$
Cauchy	$\frac{1}{\pi(1+x^2)}$	$\epsilon \sim [0;1]$	$\tan(\pi\epsilon)$
Laplace	$\mathcal{L}(0;1) = \exp(-\lvert x\rvert)$	$\epsilon \sim [0;1]$	$\ln(\frac{\epsilon_1}{\epsilon_2})$
Laplace	$\mathcal{L}(\mu;b)$	$\epsilon \sim [0;1]$	$\mu - bsgn(\epsilon)\ln(1-2\lvert\epsilon\rvert)$
Std Gaussian	$\mathcal{N}(0;1)$	$\epsilon \sim [0;1]$	$\sqrt{\ln(\frac{1}{\epsilon_1})}\cos(2\pi\epsilon_2)$
Gaussian	$\mathcal{N}(\mu;RR^\top)$	$\epsilon \sim \mathcal{N}(0;1)$	$\mu + R\epsilon$
Rademacher	$Rad(\frac{1}{2})$	$\epsilon \sim Bern(\frac{1}{2})$	$2\epsilon - 1$
Log-Normal	$\ln \mathcal{N}(\mu;\sigma)$	$\epsilon \sim \mathcal{N}(\mu;\sigma^2)$	$\exp(\epsilon)$
Inv Gamma	$i\mathcal{G}(k;\theta)$	$\epsilon \sim \mathcal{G}(k;\theta^{-1})$	$\frac{1}{\epsilon}$

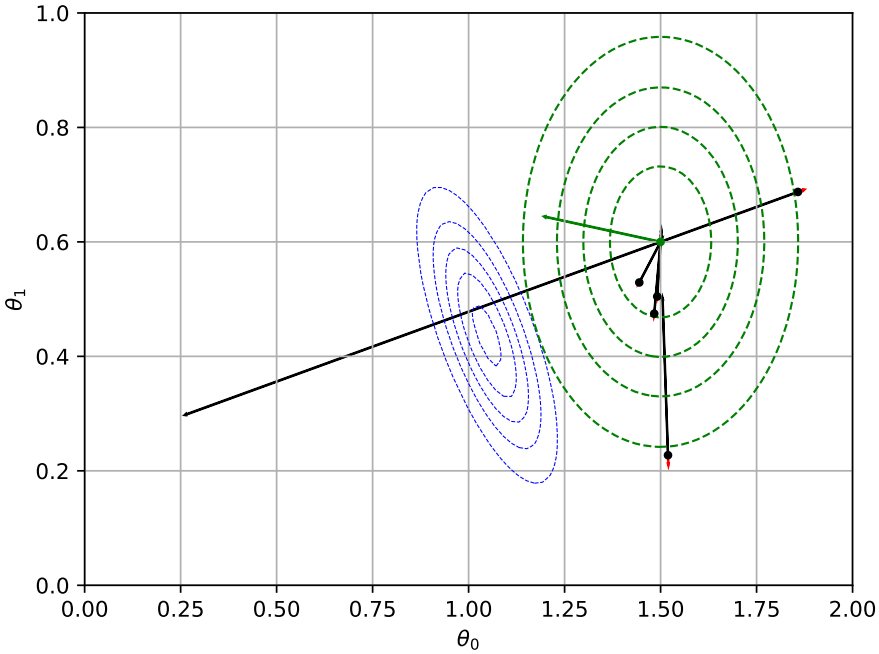
Table from http://blog.shakirm.com/2015/10/ machine-learning-trick-of-the-day-4-reparameterisation-tricks/

A nice survey (very active area of research)

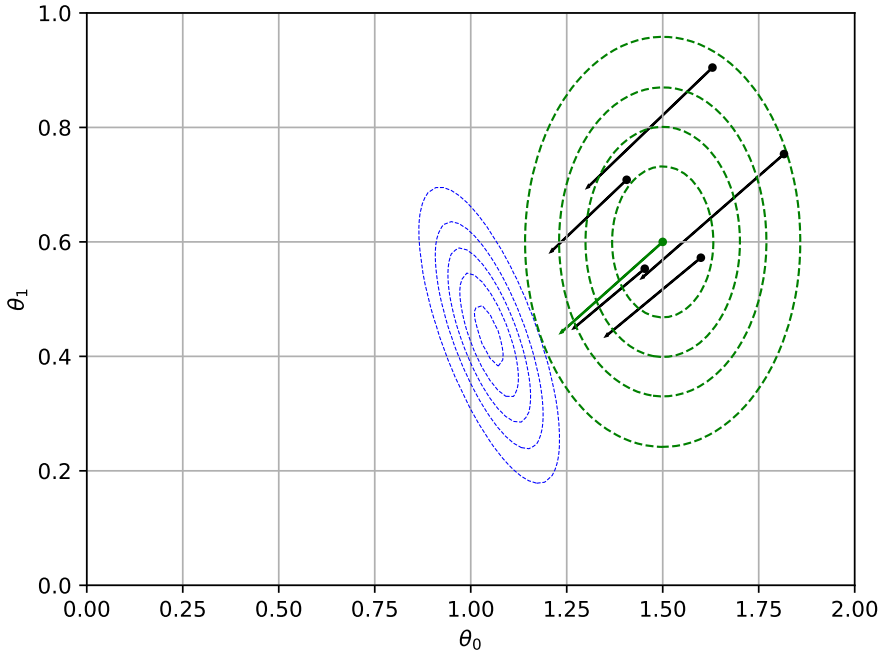
Zhang, Cheng, et al. "Advances in variational inference." IEEE transactions on pattern analysis and machine intelligence 41.8 (2018): 2008-2026.

Does it work?

Score-function gradient



Reparameterized gradient

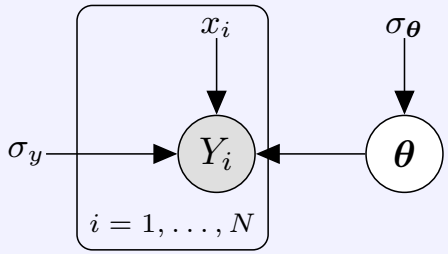


Length of gradients increased for visibility. Graphics inspired by Arto Klami @ ProbAI2021.

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 $p(\boldsymbol{\theta}|\mathcal{D})$.

Code Task: Reparameterization-gradient for linear regression



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

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

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

- We provide $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}, \mathcal{D})$, $\nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{\theta} \,|\, \boldsymbol{\lambda})$ and $\nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})$ 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.

Reparametrization: Gradients align with model's gradient ($\nabla_{\boldsymbol{\theta}} \ln p(\mathcal{D}, \boldsymbol{\theta})$). But:

Reparametrization: Gradients align with model's gradient ($\nabla_{\boldsymbol{\theta}} \ln p(\mathcal{D}, \boldsymbol{\theta})$). But:

- Requires $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ to be **reparametrizable**.
- Requires $\ln p(\mathcal{D}, \boldsymbol{\theta})$ and $\ln q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ be **differentiable** (i.e. no categorical variables).

Summary

Reparametrization: Gradients align with model's gradient ($\nabla_{\boldsymbol{\theta}} \ln p(\mathcal{D}, \boldsymbol{\theta})$). But:

- Requires $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ to be **reparametrizable**.
- Requires $\ln p(\mathcal{D}, \boldsymbol{\theta})$ and $\ln q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ be **differentiable** (i.e. no categorical variables).

Score Function: Gradients point towards the **mode of the approximation**, and the **only way the model influences them** is through $\log p(\mathcal{D}, \boldsymbol{\theta})$ in the weights.

Summary

Reparametrization: Gradients align with model's gradient ($\nabla_{\boldsymbol{\theta}} \ln p(\mathcal{D}, \boldsymbol{\theta})$). But:

- Requires $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ to be **reparametrizable**.
- Requires $\ln p(\mathcal{D}, \boldsymbol{\theta})$ and $\ln q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ be **differentiable** (i.e. no categorical variables).

Score Function: Gradients point towards the **mode of the approximation**, and the **only way the model influences them** is through $\log p(\mathcal{D}, \boldsymbol{\theta})$ in the weights.

- Only requires $\ln q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ to be **differentiable**.
- No requirements for $\ln p(\mathcal{D}, \boldsymbol{\theta})$ (only to be computable).

Summary

Reparametrization: Gradients align with model's gradient ($\nabla_{\boldsymbol{\theta}} \ln p(\mathcal{D}, \boldsymbol{\theta})$). But:

- Requires $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ to be **reparametrizable**.
- Requires $\ln p(\mathcal{D}, \boldsymbol{\theta})$ and $\ln q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ be **differentiable** (i.e. no categorical variables).

Score Function: Gradients point towards the **mode of the approximation**, and the **only way the model influences them** is through $\log p(\mathcal{D}, \boldsymbol{\theta})$ in the weights.

- Only requires $\ln q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ to be **differentiable**.
- No requirements for $\ln p(\mathcal{D}, \boldsymbol{\theta})$ (only to be computable).

Takeaway Message

Score Function is more general, but Reparametrization is better if applicable.

1. (Manual) Define your data model and the prior.

$$p(\mathcal{D}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

Automatic Variational Inference in PPLs

1. (Manual) Define your data model and the prior.

$$p(\mathcal{D}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

2. (Manual/Automatic) Define the variational distribution

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda})$$

Automatic Variational Inference in PPLs

1. (Manual) Define your data model and the prior.

$$p(\mathcal{D}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

2. (Manual/Automatic) Define the variational distribution

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda})$$

3. (Automatic) Optimize the ELBO:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \rho \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t)$$

Automatic Variational Inference in PPLs

1. (Manual) Define your data model and the prior.

$$p(\mathcal{D}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

2. (Manual/Automatic) Define the variational distribution

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda})$$

3. (Automatic) Optimize the ELBO:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \rho \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t)$$

   - Using either score-funtion or reparametrization gradients.

Automatic Variational Inference in PPLs

1. (Manual) Define your data model and the prior.

$$p(\mathcal{D}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

2. (Manual/Automatic) Define the variational distribution

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda})$$

3. (Automatic) Optimize the ELBO:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \rho \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t)$$

   - Using either score-funtion or reparametrization gradients.
   - **Automatic-Differentiation engines** take care of gradients.

Automatic Variational Inference in PPLs

1. (Manual) Define your data model and the prior.

$$p(\mathcal{D}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$$

2. (Manual/Automatic) Define the variational distribution

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda})$$

3. (Automatic) Optimize the ELBO:

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \rho \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t)$$

   - Using either score-funtion or reparametrization gradients.
   - **Automatic-Differentiation engines** take care of gradients.

4. (Automatic) Approximate inference result

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda}^\star) = \arg\min_q \mathrm{KL}\left(q(\boldsymbol{\theta}|\boldsymbol{\lambda})||p(\boldsymbol{\theta}|\mathcal{D})\right)$$

Probabilistic programming: Variational inference in Pyro

Pyro

Pyro (`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.

Simple example

$$\begin{aligned} \text{temp} &\sim \mathcal{N}(15, 2) \\ \text{sensor} &\sim \mathcal{N}(\text{temp}, 1) \end{aligned}$$

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

Simple example

$$\begin{aligned} \text{temp} &\sim \mathcal{N}(15, 2) \\ \text{sensor} &\sim \mathcal{N}(\text{temp}, 1) \end{aligned}$$

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

Pyro models:

- random variables $\Leftrightarrow$ `pyro.sample`
- observations $\Leftrightarrow$ `pyro.sample` with the `obs` argument

Simple example

$$\begin{aligned} \text{temp} &\sim \mathcal{N}(15, 2) \\ \text{sensor} &\sim \mathcal{N}(\text{temp}, 1) \end{aligned}$$

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

Pyro models:

- random variables $\Leftrightarrow$ `pyro.sample`
- observations $\Leftrightarrow$ `pyro.sample` with the `obs` argument

```
#The observatons
obs = {'sensor': torch.tensor(18.0)}

def model(obs):
    temp = pyro.sample('temp', dist.Normal(15.0, 2.0))
    sensor = pyro.sample('sensor', dist.Normal(temp, 1.0), obs=obs['sensor'])
```

Inference Problem

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

Inference Problem

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

Variational Solution

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

Inference Problem

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

Variational Solution

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

Pyro Guides:

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

Inference Problem

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

Variational Solution

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

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

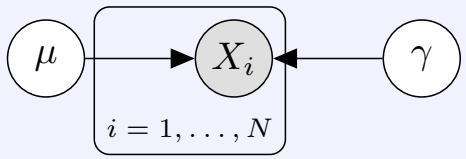
```
#The observatons
obs = {'sensor': torch.tensor(18.0)}

def model(obs):
    temp = pyro.sample('temp', dist.Normal(15.0, 2.0))
    sensor = pyro.sample('sensor', dist.Normal(temp, 1.0), obs=obs['sensor'])
```

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

Exercise: Pyro implementation for a simple Gaussian model

`Day2-AfterLunch/student_simple_gaussian_model_pyro.ipynb`



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

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

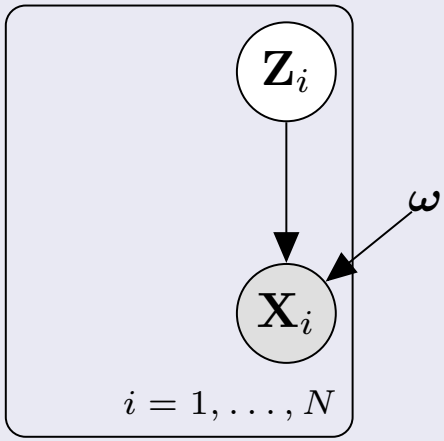
$$q(\mu, \gamma) = ...$$

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

`Day2-BeforeLunch/student_simple_model.ipynb`

Deep Bayesian Learning – VAE

Model of interest

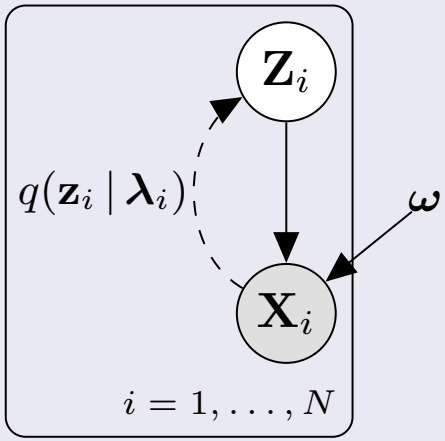


- $p(\mathbf{z}_i)$ is (usually) an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))$, where g is a deep neural network.

$$p_{\boldsymbol{\omega}}(\mathbf{x}_i|\mathbf{z}_i) \sim \text{Bernoulli}(\text{logits} = g_{\boldsymbol{\omega}}(\mathbf{z}_i))$$

- $g_{\boldsymbol{\omega}}(\mathbf{z}_i)$ plays the role of a **DECODER NETWORK**.
- Learn $\boldsymbol{\omega}$ to maximize the model's fit to $\mathcal{D}$.
  - We will cheat and find a **point estimate** for $\boldsymbol{\omega}$.

Model of interest



- $p(\mathbf{z}_i)$ is (usually) an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))$, where g is a deep neural network.

$$p_{\boldsymbol{\omega}}(\mathbf{x}_i|\mathbf{z}_i) \sim \text{Bernoulli}(\text{logits} = g_{\boldsymbol{\omega}}(\mathbf{z}_i))$$

- $g_{\boldsymbol{\omega}}(\mathbf{z}_i)$ plays the role of a **DECODER NETWORK**.
- Learn $\boldsymbol{\omega}$ to maximize the model's fit to $\mathcal{D}$.
  - We will cheat and find a **point estimate** for $\boldsymbol{\omega}$.

Variational Inference

- We will need $p_{\boldsymbol{\omega}}(\mathbf{z}_i \mid \mathbf{x}_i)$ for each data-point $\mathbf{x}_i$:

$$p_{\boldsymbol{\omega}}(\mathbf{z}_i \mid \mathbf{x}_i) = \frac{p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))}{\int_{\mathbf{z}_i} p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i)) \, \mathrm{d}\mathbf{z}_\mathrm{i}}.$$

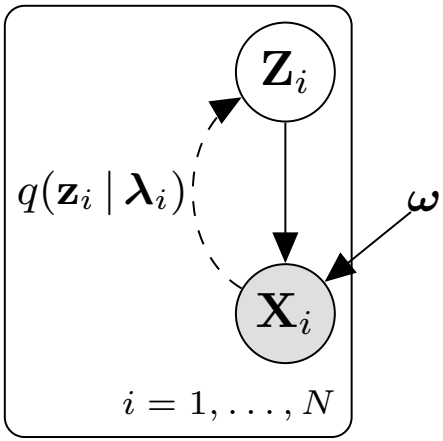
- **Initial plan:** Fit $q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)$ to $p_{\boldsymbol{\omega}}(\mathbf{z}_i \mid \mathbf{x}_i)$ using variational inference.

Initial plan:

- Optimize the ELBO

$$\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_N) = -\mathbb{E}_q \left[\log \frac{\prod_{i=1}^N q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)}{\prod_{i=1}^N p_{\boldsymbol{\omega}}(\mathbf{z}_i, \mathbf{x}_i)} \right].$$

- A natural model for $q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)$ is a Gaussian with parameters $\boldsymbol{\lambda}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$.
- If $\mathbf{Z}_i$ is d-dim and we for simplicity assume diagonal $\boldsymbol{\Sigma}_i$, this still gives $2Nd$ variational parameters to learn.

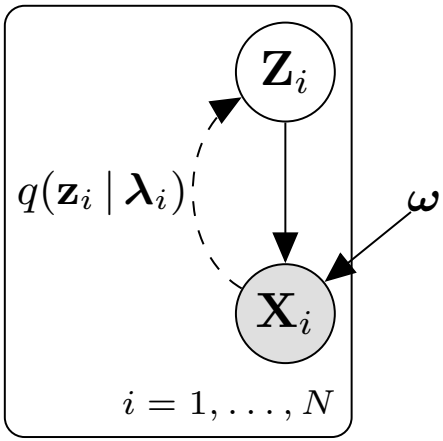


Initial plan:

- Optimize the ELBO

$$\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_N) = -\mathbb{E}_q \left[\log \frac{\prod_{i=1}^N q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)}{\prod_{i=1}^N p_{\boldsymbol{\omega}}(\mathbf{z}_i, \mathbf{x}_i)} \right].$$

- A natural model for $q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)$ is a Gaussian with parameters $\boldsymbol{\lambda}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$.
- If $\mathbf{Z}_i$ is d-dim and we for simplicity assume diagonal $\boldsymbol{\Sigma}_i$, this still gives $2Nd$ variational parameters to learn.



A better plan

- Assume $g_{\boldsymbol{\omega}}(\mathbf{z})$ is "smooth": if $\mathbf{z}_i$ and $\mathbf{z}_j$ are "close", then so are $\mathbf{x}_i$ and $\mathbf{x}_j$.

  $\rightsquigarrow$ $\boldsymbol{\lambda}_i$ and $\boldsymbol{\lambda}_j$ should be "close" if $\mathbf{x}_i$ and $\mathbf{x}_j$ are "close".

- **Therefore:** Let's assume there exists a (smooth) function $h(\mathbf{x})$ so that $h(\mathbf{x}_i) = \boldsymbol{\lambda}_i$.
- $h(\cdot)$ is unavailable, so represent it using a deep neural net and learn the weights.
- $h(\mathbf{x}_i)$ plays the role of an **ENCODER NETWORK**.

Amortized inference:
To learn a model $h(\cdot)$, typically a deep neural network, so that $h(\mathbf{x}_i) = \boldsymbol{\lambda}_i$.
$h(\cdot)$ is parameterized with weights, often (abusing notation) denoted by $\boldsymbol{\lambda}$.

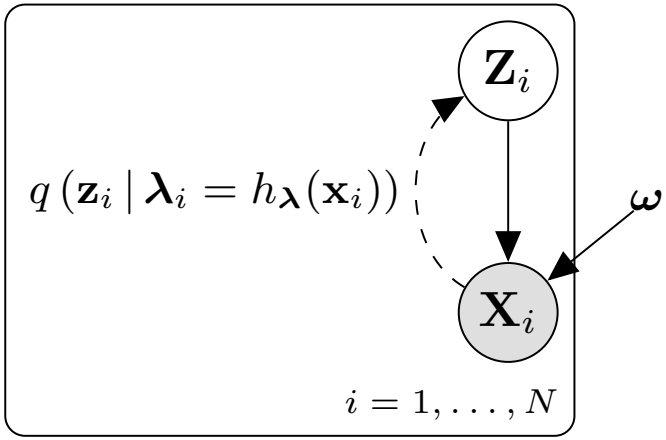
Note! Amortized inference is useful also outside VAEs!

Benefits:

- The $2Nd$ parameters $\{\boldsymbol{\lambda}_i\}_{i=1}^N$ are replaced by the fixed-sized vector $\boldsymbol{\lambda}$.
  - If N is large we may get a simpler learning problem.
- Smoothness of $h(\cdot)$ implies regularization.
- We only change the **parameterization**, not the model itself!

The full VAE approach:

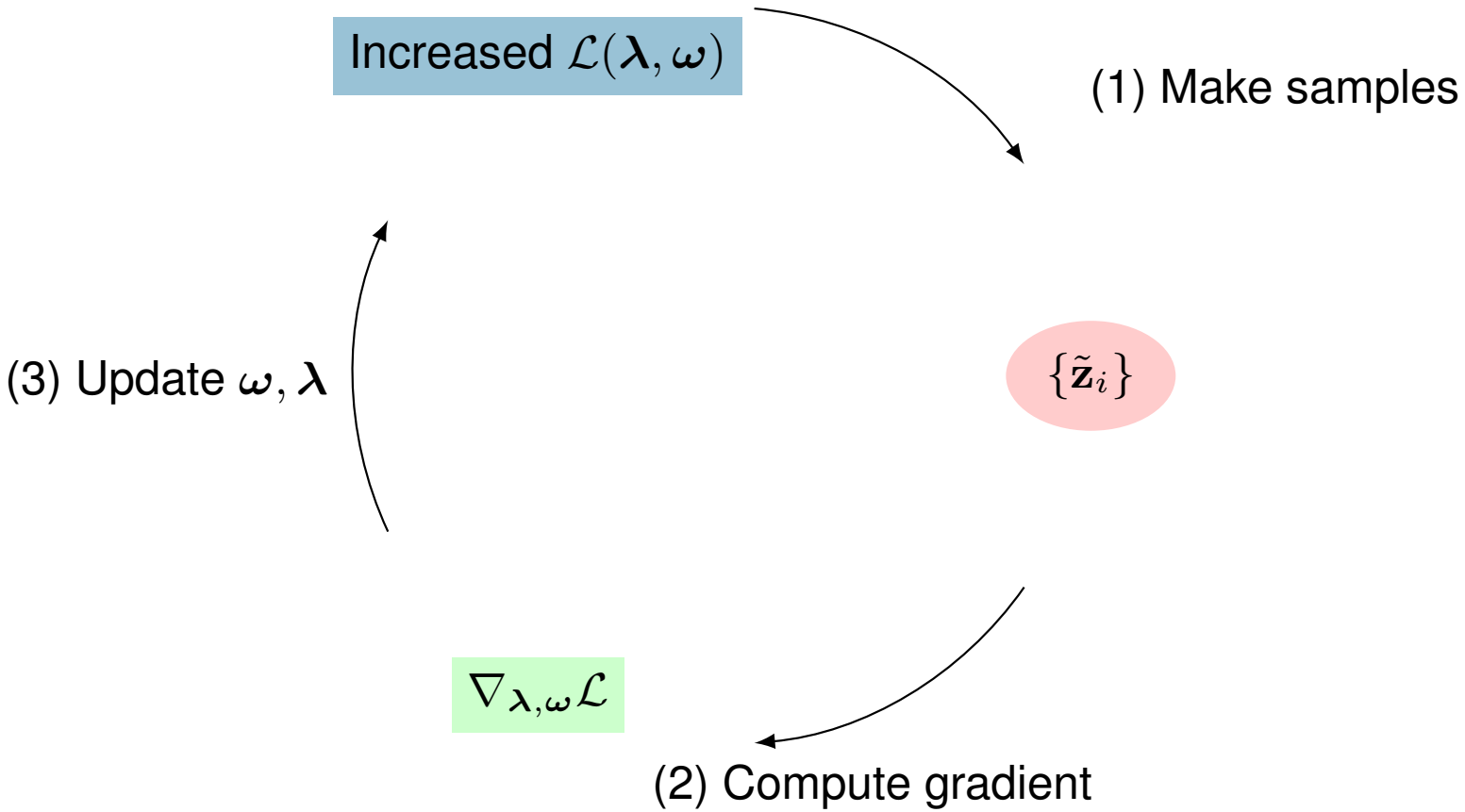
- $p(\mathbf{z}_i)$ is an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i|\mathbf{z}_i) \sim$ Bernoulli(logits $= g_{\boldsymbol{\omega}}(\mathbf{z}_i)$), where $g_{\boldsymbol{\omega}}$ is a DNN with weights $\boldsymbol{\omega}$.
- $q(\mathbf{z}_i \,|\, \mathbf{x}_i, \boldsymbol{\lambda}) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where $\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ is given by $h_{\boldsymbol{\lambda}}(\mathbf{x}_i)$. $h_{\boldsymbol{\lambda}}$ is a DNN with weights $\boldsymbol{\lambda}$.



Goal:

Learn **both** $\boldsymbol{\omega}$ and $\boldsymbol{\lambda}$ by maximizing the ELBO:

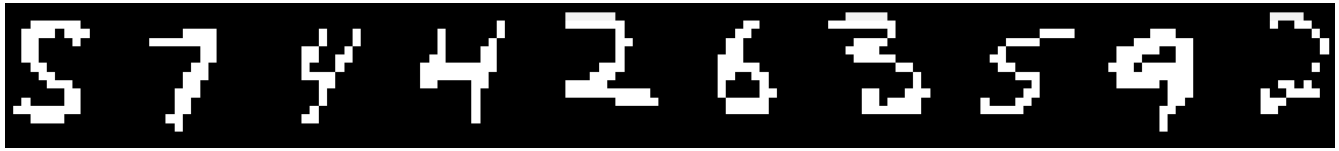
$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\omega}) = -\mathbb{E}_q \left[\log \frac{q(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\lambda})}{p_{\boldsymbol{\omega}}(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\omega})} \right].$$



1. For each $\mathbf{x}_i$, sample M (typically 1) $\boldsymbol{\epsilon}$-values.
2. Calculate $\nabla_{\boldsymbol{\lambda},\boldsymbol{\omega}}\, \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\omega})$ using the reparameterization-trick.
3. Update parameters using a standard DL optimizer (like Adam).

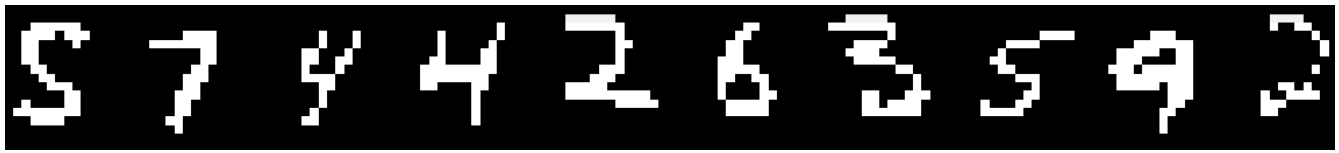
Fun with MNIST – The model

- The model is learned from $N = 55.000$ training examples.
- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit ("0" – "9").

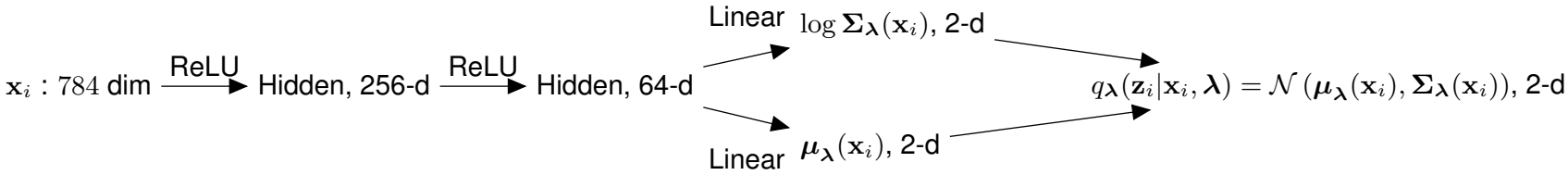


Fun with MNIST – The model

- The model is learned from $N = 55.000$ training examples.
- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit ("0" – "9").

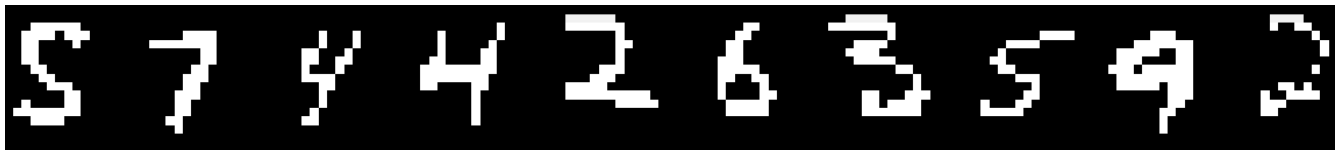


- Encoding is done in **two** dimensions. $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$.



Fun with MNIST – The model

- The model is learned from $N = 55.000$ training examples.
- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit ("0" – "9").



- Encoding is done in **two** dimensions. $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$.
- The **decoder network** $\mathbf{Z} \rightsquigarrow \mathbf{X}$ is a $64 + 256$ neural net with ReLU units.

$$\mathbf{z}_i : 2 \text{ dim} \xrightarrow{\text{ReLU}} \text{Hidden, 64-d} \xrightarrow{\text{ReLU}} \text{Hidden, 256-d} \xrightarrow{\text{Linear}} \text{logit}(\mathbf{p}_i), \text{ 784-d} \longrightarrow p_{\boldsymbol{\omega}}(\mathbf{x}_i \,|\, \mathbf{z}_i, \boldsymbol{\omega}) = \text{Bernoulli}\,(\mathbf{p}_i), \text{ 784-d}$$

Code Task: VAEs in Pyro

- Learn how a VAE is coded in Pryo.
- We provide a VAE with a **linear decoder**.
- **Exercise (summary):**
  - Define a Non-Linear Decoder, e.g., an MLP with a hidden layer and non-linearities (e.g. Relu).
  - Explore the latent space when moving from linear to non-linear decoders with different capacity.
- Notebook:

`Day2-AfterLunch/students_VAE.ipynb`.

Conclusions

- **Bayesian Machine Learning**
  - Represents unobserved quantities using **distributions**
  - Models **epistemic** uncertainty using $p(\boldsymbol{\theta} \mid \mathcal{D})$

- **Bayesian Machine Learning**

- **Variational inference**
  - **Provides** $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$**:** A distributional approximation to $p(\boldsymbol{\theta} \mid \mathcal{D})$
  - **Objective:** $\arg\min_{\boldsymbol{\lambda}} \mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) || p(\boldsymbol{\theta} \mid \mathcal{D})\right) \Leftrightarrow \arg\max_{\boldsymbol{\lambda}} \mathcal{L}\left(q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})\right)$
  - **Mean-field:** Divide and conquer strategy for high-dimensional posteriors
  - **Main caveat:** $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$ underestimates the uncertainty of $p(\boldsymbol{\theta} \mid \mathcal{D})$

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
  - Analytic expressions for some models (i.e., conjugate exponential family)
  - CAVI is very **efficient and stable** if it can be used
  - In principle requires **manual derivation** of updating equations
    - There are **tools** to help (using *variational message passing*)

Recap & Conclusions

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
- **Gradient-based Variational Inference**
  - Provides the tools for VI over **arbitrary** probabilistic models
  - Directly integrates with the tools of deep learning
    - Automatic differentiation, sampling from standard distributions, and SGD
  - Sampling to approximate expectations: **Beware of the variance!**

Recap & Conclusions

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
- **Gradient-based Variational Inference**
- **Probabilistic programming languages**
  - PPLs fuel the “build – compute – critique – repeat” - cycle through
    - ease and flexibility of modelling
    - powerful inference engines
    - efficient model evaluations
  - Many available tools (Pyro, TF Probability, Infer.net, Turing.jl, . . .)

Recap & Conclusions

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
- **Gradient-based Variational Inference**
- **Probabilistic programming languages**
- **What's next?**
  - The "VI toolbox" is reaching maturity
    - From *only* a research area to almost a *prerequisite* for Probabilistic AI
    - ... yet there are still things to explore further!
  - Today's material should suffice to read (and write!) Prob-AI papers