Tropical Probabilistic AI school

Variational Inference and Optimization

Helge Langseth, Bjørn Magnus Mathisen, and Eliezer de Souza da Silva

This material is very heavily based on what was prepared for ProbAI-23 by
Helge Langseth, Andrés Masegosa, and Thomas Dyhre Nielsen

January 30, 2024

Demo: Bayesian Neural Network

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

. . . 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

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:
  - $\sum_t \rho_t^2 < \infty$
  - $\sum_t \rho_t = \infty$

Black Box Variational Inference

Main idea: Cast inference as an optimization problem

Optimize the ELBO by stochastic gradient ascent over the parameters $\boldsymbol{\lambda}$. 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

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

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 $q(\theta_i \mid \boldsymbol{\lambda}_i)$.

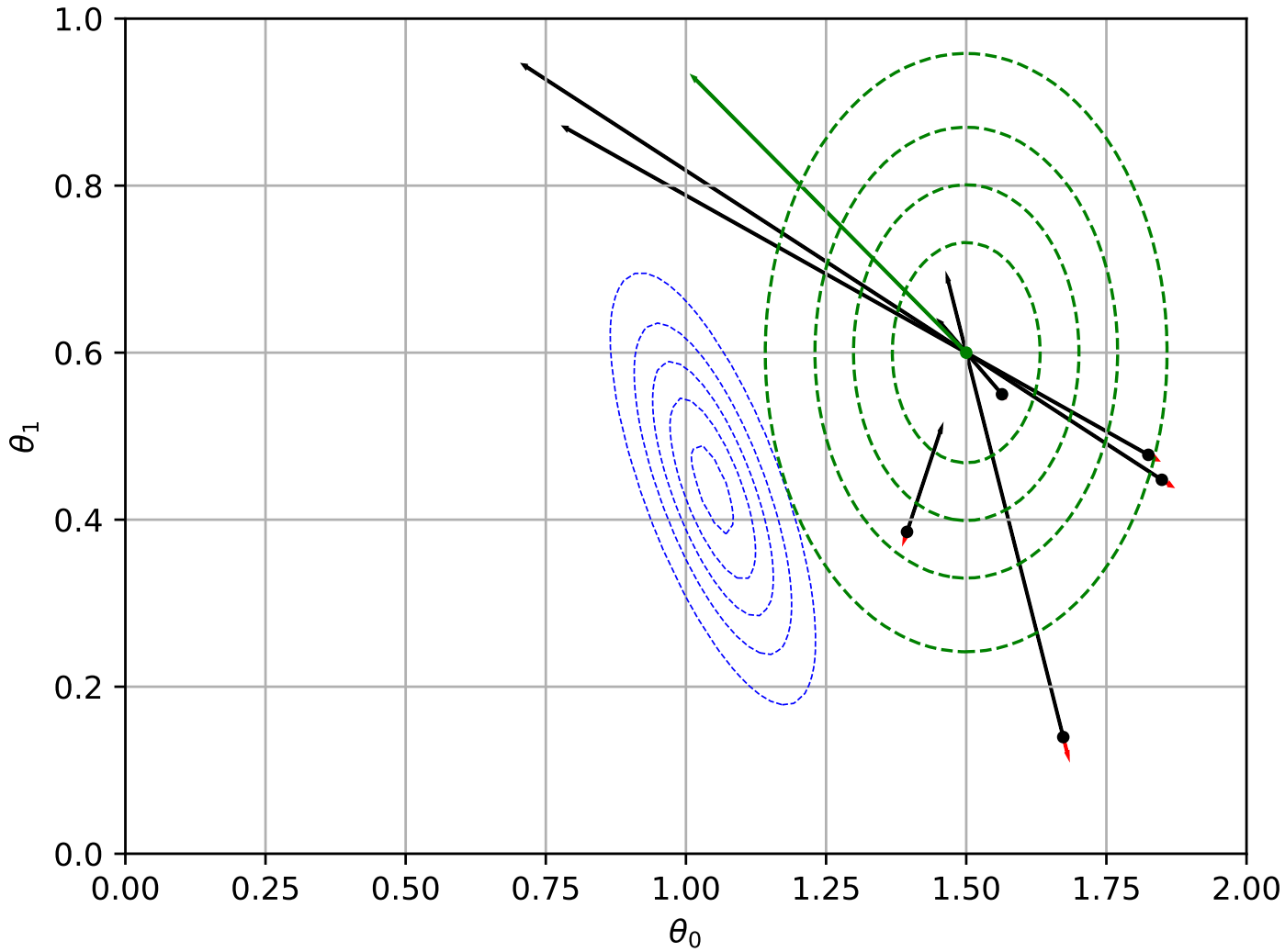
Calculating the gradient – in summary

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 fairly small.

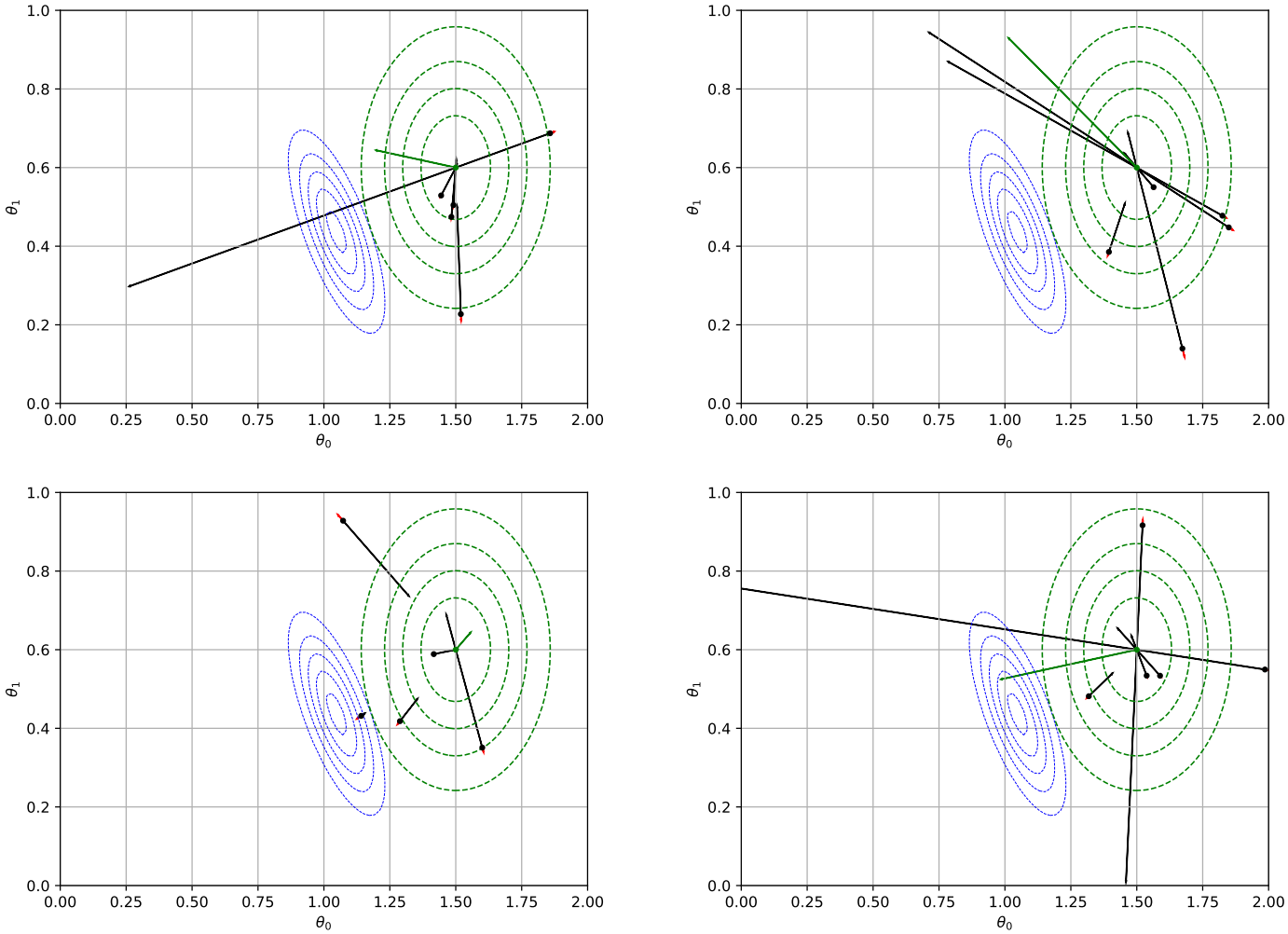
Does it work?



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

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

Does it work?

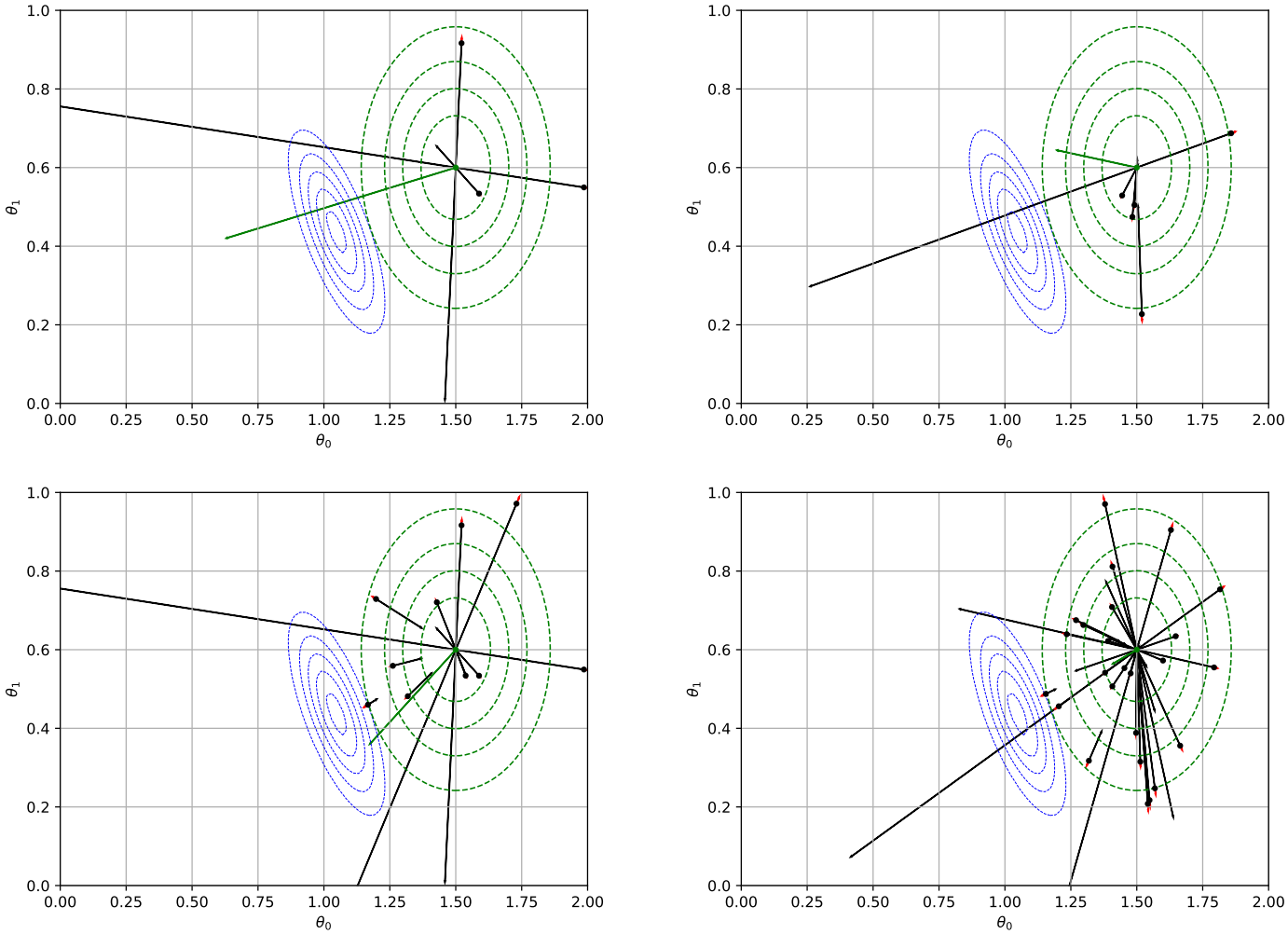


Different samples, each with $M = 5$.

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

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

Does it work?

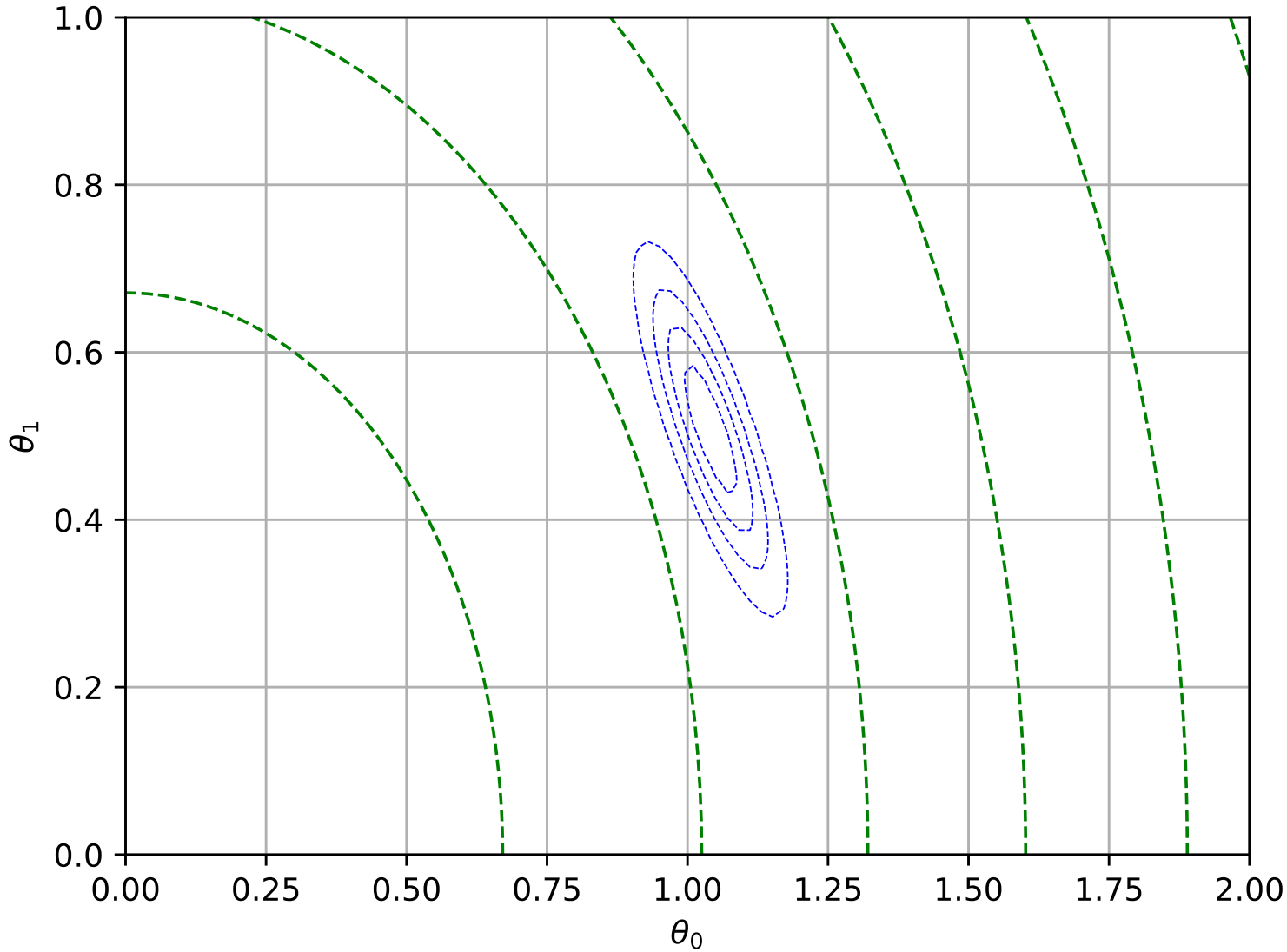


Different values of M ($M = 3$, 5, 10, and 25)

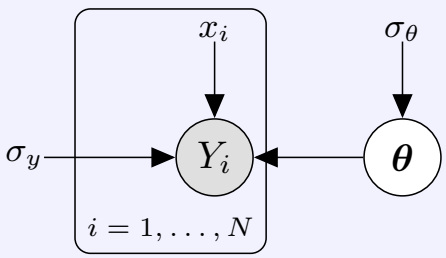
$\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\log \frac{p(\boldsymbol{\theta}_i, \mathcal{D})}{q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})} \cdot \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}_i \mid \boldsymbol{\lambda})$; $\frac{1}{M} \sum_{j=1}^{M}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

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



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

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

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

- Look at `Exercise 1` in the notebook `Day2/students_BBVI.ipynb`.
- Calculate $\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta} \,|\, \boldsymbol{\lambda})$, 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

Goal: Find a more robust estimator for the gradient

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

The Reparametrization Trick

Goal: Find a more robust estimator for the gradient

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

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

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

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

The Reparametrization Trick

Goal: Find a more robust estimator for the gradient

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

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

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

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

Example: The Gaussian distribution

Assume $q(\boldsymbol{\theta}|\boldsymbol{\lambda}) = \mathcal{N}(\boldsymbol{\theta} \mid \boldsymbol{\mu}, \mathbf{R}\mathbf{R}^\mathsf{T})$, so $\boldsymbol{\lambda} = \{\boldsymbol{\mu}, \mathbf{R}\}$. $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ can be reparametrized using

$$\begin{aligned} \boldsymbol{\epsilon} \sim \phi(\boldsymbol{\epsilon}) \quad &= \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ \boldsymbol{\theta} = f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}) &= \boldsymbol{\mu} + \mathbf{R}\,\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{R}\mathbf{R}^\mathsf{T}). \end{aligned}$$

Univariate case: $q(\theta|\boldsymbol{\lambda}) = \mathcal{N}(\mu, \sigma^2)$ gives $\epsilon \sim \mathcal{N}(0, 1)$ and $\theta = f(\epsilon, \boldsymbol{\lambda}) = \mu + \sigma\,\epsilon$.

The Reparametrization Trick

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

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

Consequences for how we can calculate the gradient:

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

The Reparametrization Trick

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

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

Consequences for how we can calculate the gradient:

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

The Reparametrization Trick

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

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

Consequences for how we can calculate the gradient:

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

The Reparametrization Trick

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

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

Consequences for how we can calculate the gradient:

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

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

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

Consequences for how we can calculate the gradient:

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{\boldsymbol{\theta} \sim q(\cdot\,|\,\boldsymbol{\lambda})} \left[\log \frac{p(\,\boldsymbol{\theta}\,, \mathcal{D})}{q(\,\boldsymbol{\theta}\,|\,\boldsymbol{\lambda})} \right] \\ &= \nabla_{\boldsymbol{\lambda}} \mathbb{E}_{\boldsymbol{\epsilon} \sim \phi(\cdot)} \left[\log \frac{p(\,f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\,, \mathcal{D})}{q(\,f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\,|\,\boldsymbol{\lambda})} \right] \\ &= \mathbb{E}_{\boldsymbol{\epsilon} \sim \phi} \left[\nabla_{\boldsymbol{\lambda}} \log \frac{p(f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}), \mathcal{D})}{q(f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\,|\,\boldsymbol{\lambda})} \right] \\ &= \mathbb{E}_{\boldsymbol{\epsilon} \sim \phi} \left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta}\,|\,\boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}) - \nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda}) \right] \\ &= \mathbb{E}_{\boldsymbol{\epsilon} \sim \phi} \left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta}\,|\,\boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda}) \right] - 0 \end{aligned}$$

(See Slide 6, Point 3)

Monte-Carlo Estimation:

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

Monte-Carlo Estimation:

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

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

Monte-Carlo Estimation:

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

Monte-Carlo Estimation:

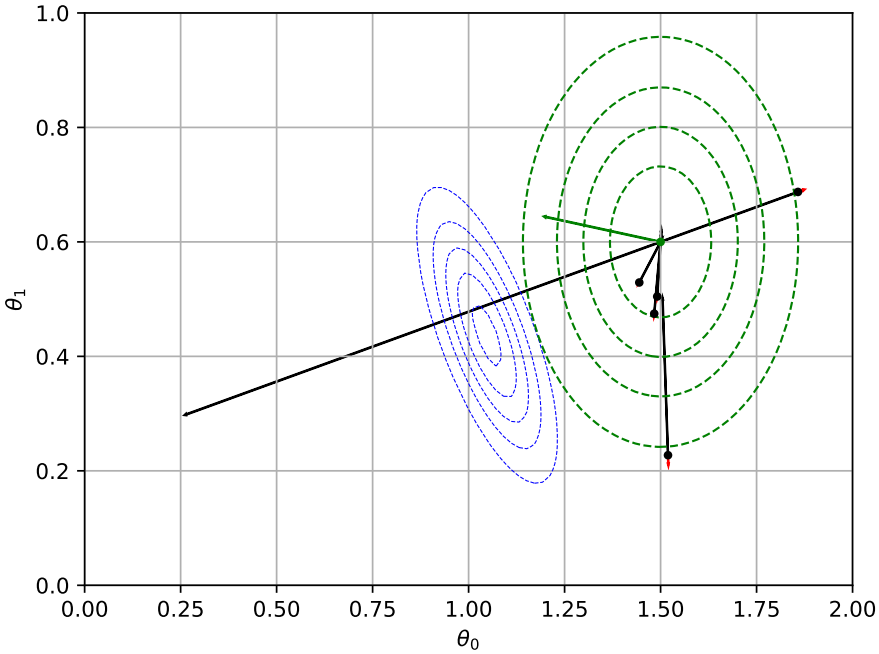
$$
\begin{aligned}
\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) &= \mathbb{E}_{\boldsymbol{\epsilon} \sim \phi}\left[\nabla_{\boldsymbol{\theta}} \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})} \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})\right]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

This gradient estimator...

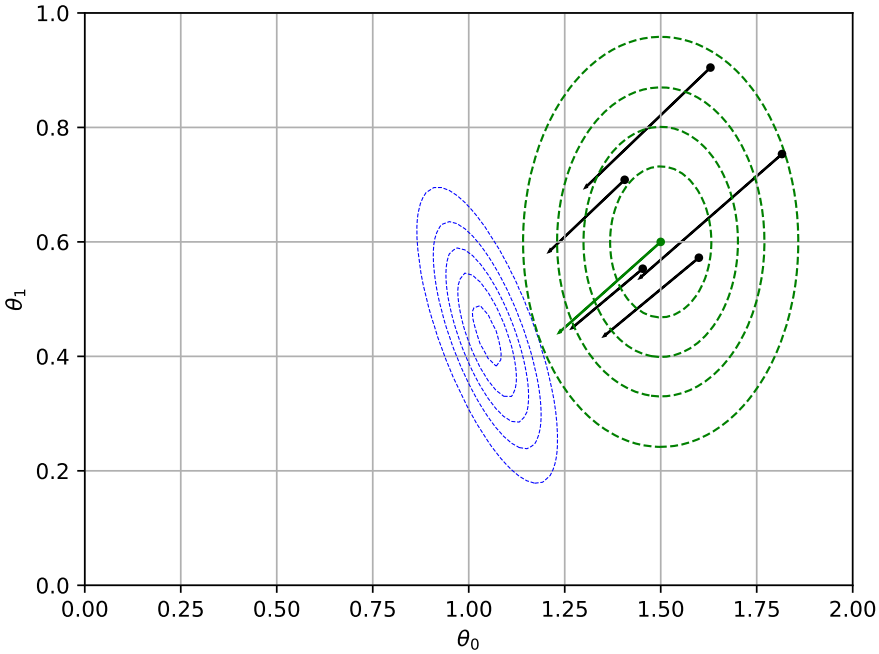
- Uses the *model's* gradients (not so for the score-function gradient).
- Requires $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ to be *reparametrizable* and *differentiable*.
- Requires $\log p(\boldsymbol{\theta}, \mathcal{D})$ to be differentiable wrt. $\boldsymbol{\theta}$.

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

Score function gradients:

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

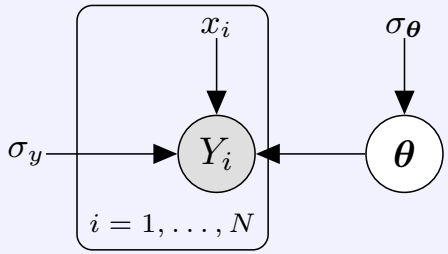
Reparametrization gradients:

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

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



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

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

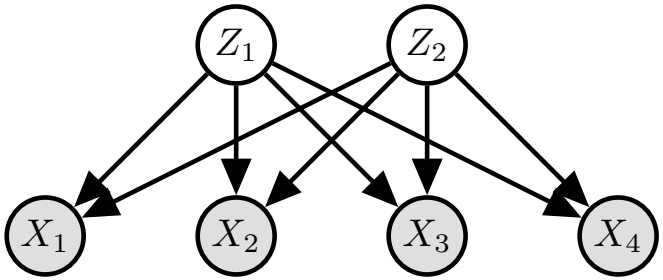
$$\nabla_{\boldsymbol{\lambda}} \mathcal{L}(q) = \mathop{\mathbb{E}}_{\epsilon \sim \phi} \left[\left(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}, \mathcal{D}) - \nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{\theta} \,|\, \boldsymbol{\lambda}) \right) \nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon},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

- 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/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.

Deep Bayesian Learning – VAE

Starting-point: The factor analysis model, and an extension

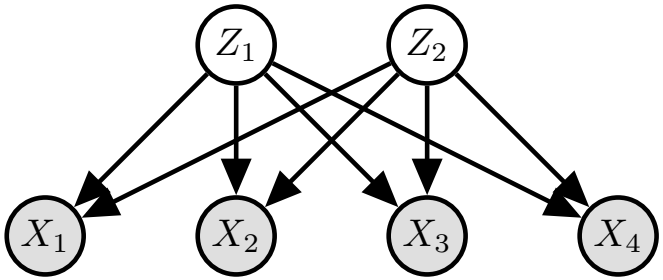


$$\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\mathbf{X} \,|\, \mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z}, \boldsymbol{\Sigma})$$

- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \,|\, \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \,|\, \mathbf{x})$ (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse $\mathbf{W}$ eases the interpretation.

Starting-point: The factor analysis model, and an extension



$$\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\mathbf{X} \,|\, \mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z}, \boldsymbol{\Sigma})$$

- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \,|\, \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \,|\, \mathbf{x})$ (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse $\mathbf{W}$ eases the interpretation.

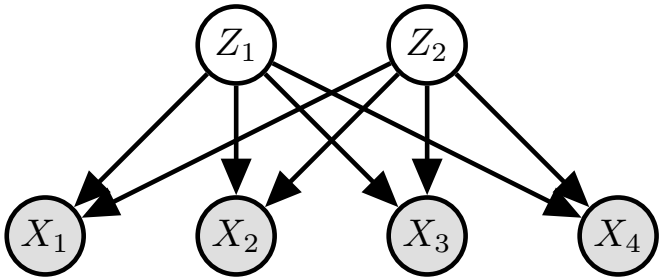
Example: Grades

We observe $\mathbf{x} = \{\texttt{Math}, \texttt{English}, \texttt{Computer Science}, \texttt{German}\}$ for N students, and will examine the data with an FA. Say the model gives us

$$\mathbb{E}[\mathbf{Z} \,|\, \mathbf{x}] = \begin{bmatrix} .25 & .25 & .25 & .25 \\ .50 & 0 & .35 & .15 \end{bmatrix} \cdot \begin{bmatrix} \texttt{Math} \\ \texttt{English} \\ \texttt{Computer Science} \\ \texttt{German} \end{bmatrix}$$

Possible interpretation: $Z_1 \approx$ “Eagerness to learn” and $Z_2 \approx$ “Logical thinking”.

Starting-point: The factor analysis model, and an extension



$$\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\mathbf{X} \,|\, \mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z}, \boldsymbol{\Sigma})$$

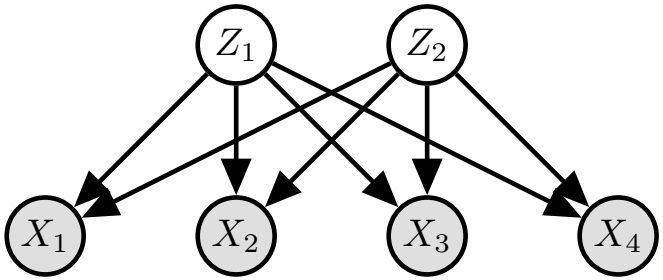
- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \,|\, \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \,|\, \mathbf{x})$ (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse $\mathbf{W}$ eases the interpretation.

How do we feel about the FA model?

The good: Data is compressed into a (hopefully) interpretable low-dimensional representation.

The bad: The model is restrictive: Assumes everything is Gaussian, and that the relationship from $\mathbf{Z}$ to $\mathbf{X}$ has to be linear.

Starting-point: The factor analysis model, and an extension



VAE: $\mathbf{Z} \sim$ "Whatever", typically still $\mathcal{N}(\mathbf{0}, \mathbf{I})$

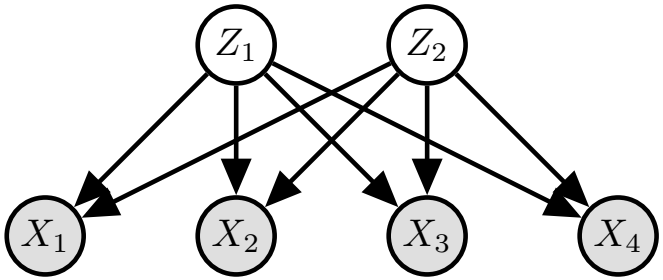
VAE: $\mathbf{X} \mid \mathbf{z} \sim$ "Whatever"

- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \mid \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \mid \mathbf{x})$ (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse $\mathbf{W}$ eases the interpretation.

From Factor Analysis to Variational Auto Encoders

VAEs allow the distribution $p(\mathbf{x} \mid \mathbf{z})$ to be **arbitrarily complex** – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate $p(\mathbf{z} \mid \mathbf{x})$, and it is therefore harder to interpret the factors $\mathbf{Z}$.

Starting-point: The factor analysis model, and an extension



VAE: $\mathbf{Z} \sim$ "Whatever", typically still $\mathcal{N}(\mathbf{0}, \mathbf{I})$

VAE: $\mathbf{X} \mid \mathbf{z} \sim$ "Whatever"

- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \mid \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \mid \mathbf{x})$ (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse $\mathbf{W}$ eases the interpretation.

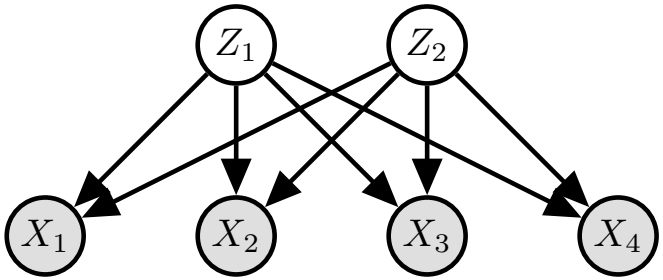
From Factor Analysis to Variational Auto Encoders

VAEs allow the distribution $p(\mathbf{x} \mid \mathbf{z})$ to be **arbitrarily complex** – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate $p(\mathbf{z} \mid \mathbf{x})$, and it is therefore harder to interpret the factors $\mathbf{Z}$.

Why that name?

VAEs are called **auto-encoders** because we can train them by "re-creating" inputs via the process $\mathbf{x} \overset{p(\mathbf{z} \mid \mathbf{x})}{\rightsquigarrow} \mathbf{z} \overset{p(\mathbf{x} \mid \mathbf{z})}{\rightsquigarrow} \hat{\mathbf{x}}$ (and expect to see $\mathbf{x} \approx \hat{\mathbf{x}}$).

Starting-point: The factor analysis model, and an extension



VAE: $\mathbf{Z} \sim$ "Whatever", typically still $\mathcal{N}(\mathbf{0}, \mathbf{I})$

VAE: $\mathbf{X} \mid \mathbf{z} \sim$ "Whatever"

- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \mid \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\mathsf{T}\mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \mid \mathbf{x})$ (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse $\mathbf{W}$ eases the interpretation.

From Factor Analysis to Variational Auto Encoders

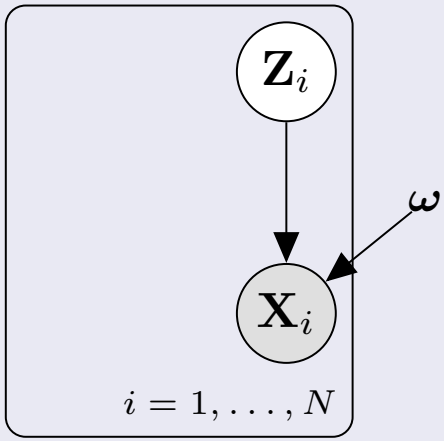
VAEs allow the distribution $p(\mathbf{x} \mid \mathbf{z})$ to be **arbitrarily complex** – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate $p(\mathbf{z} \mid \mathbf{x})$, and it is therefore harder to interpret the factors $\mathbf{Z}$.

Why that name?

VAEs are called **auto-encoders** because we can train them by "re-creating" inputs via the process $\mathbf{x} \overset{p(\mathbf{z} \mid \mathbf{x})}{\rightsquigarrow} \mathbf{z} \overset{p(\mathbf{x} \mid \mathbf{z})}{\rightsquigarrow} \hat{\mathbf{x}}$ (and expect to see $\mathbf{x} \approx \hat{\mathbf{x}}$).

It is a **variational** auto-encoder since we use the variational objective while learning.

Model of interest

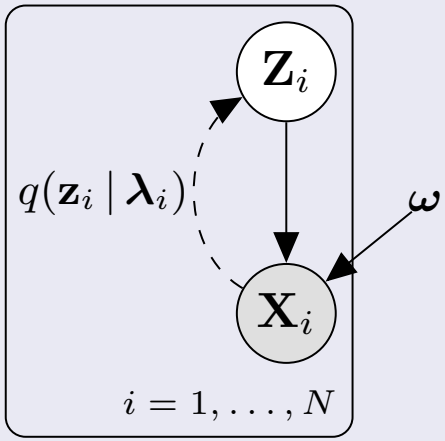


- $p(\mathbf{z}_i)$ is (usually) an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))$, where g is a deep neural network.

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

- $g_{\boldsymbol{\omega}}(\mathbf{z}_i)$ 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 $\mathbf{x}_i$:

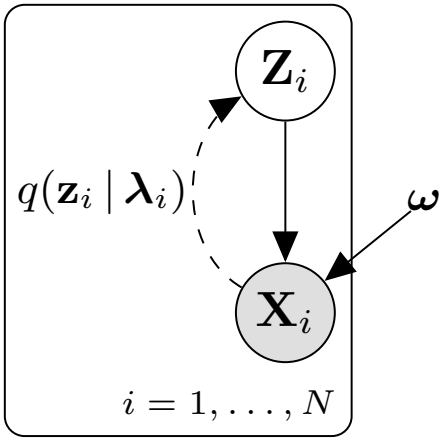
$$p_{\boldsymbol{\omega}}(\mathbf{z}_i \mid \mathbf{x}_i) = \frac{p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))}{\int_{\mathbf{z}_i} p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))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

- **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

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

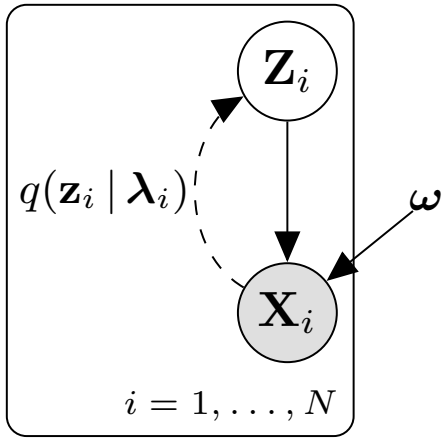


- A natural model for $q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)$ is a Gaussian with parameters $\boldsymbol{\lambda}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$.
- If $\mathbf{Z}_i$ is d-dim and we for simplicity assume diagonal $\boldsymbol{\Sigma}_i$, this still gives $2Nd$ variational parameters to learn.
- An $\tilde{\mathbf{x}} \notin \mathcal{D}$ at query time will be problematic.

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.
- An $\tilde{\mathbf{x}} \notin \mathcal{D}$ at query time will be problematic.

A better plan

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

  $\rightsquigarrow$ If $\mathbf{x}_i$ and $\mathbf{x}_j$ are "close", then $\boldsymbol{\lambda}_i$ and $\boldsymbol{\lambda}_j$ should be "close", too.

- **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 $\boldsymbol{\lambda}$.

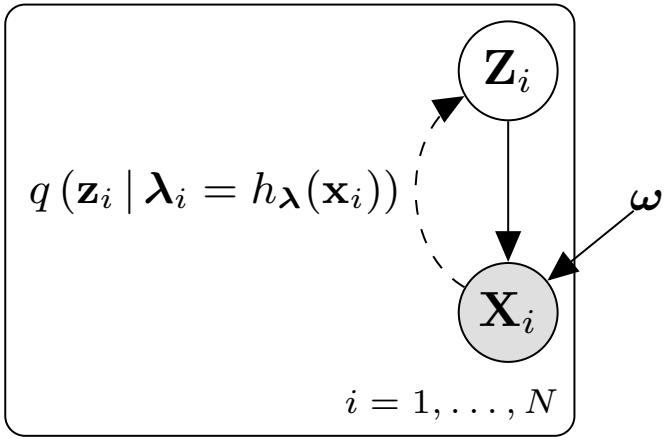
Note! Amortized inference is useful also outside VAEs!

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 $\boldsymbol{\lambda}$.

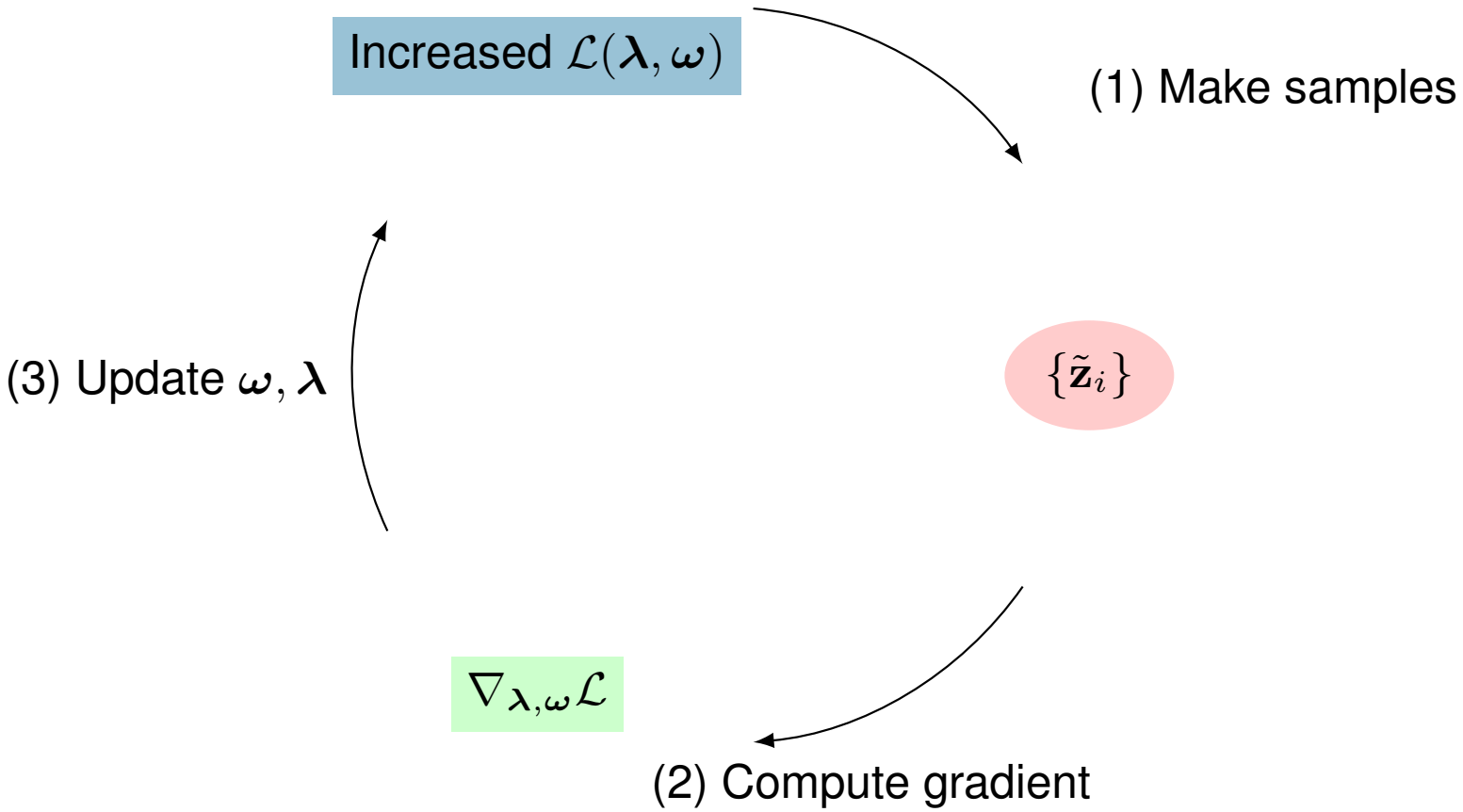


Goal:

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

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

ELBO for VAEs



1. For each $\mathbf{x}_i$, sample M (typically $M = 1$) $\boldsymbol{\epsilon}$-values to get $\mathbf{z}_i$ samples.
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).

Sidestep: Automatic Variational Inference in PPLs

1. Manual : Define your data model $p(\mathcal{D}|\boldsymbol{\theta})$ and the prior $p(\boldsymbol{\theta})$.
2. Automatic : Define the set of variational distributions $q(\boldsymbol{\theta}|\boldsymbol{\lambda}) \in \mathcal{Q}$.
   (In *complicated* situations this step may have to be Manual).
3. Automatic : Optimize ELBO: $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \rho \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t)$ using an AutoDiff. engine.
4. Automatic : Find $q(\boldsymbol{\theta}|\boldsymbol{\lambda}^\star) = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\left(q(\boldsymbol{\theta}|\boldsymbol{\lambda}) || p(\boldsymbol{\theta}|\mathcal{D})\right)$.

Probabilistic Programming Languages and Box's loop

Modern PPLs relieve us of all the computational details!

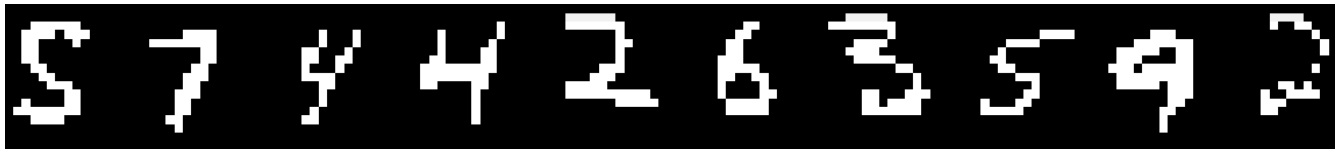
Instead we can focus on . . .

- Building models (define $p(\mathcal{D}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$) we believe in.
- Using computed results to validate/critique and iteratively refine the model.

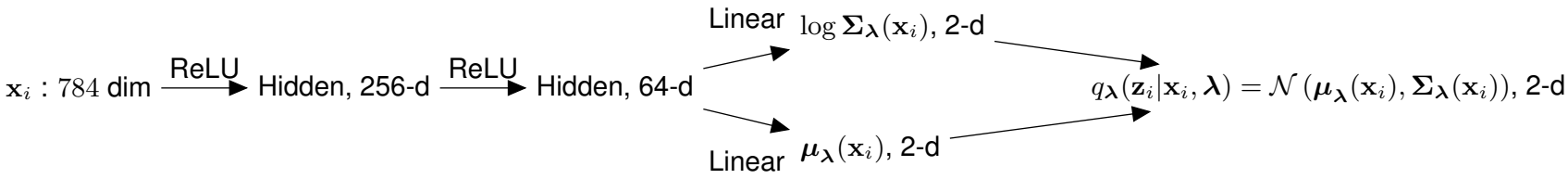
This is known as the "build – compute – critique – repeat" - cycle.

Fun with MNIST – With simple model evaluation

- 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").

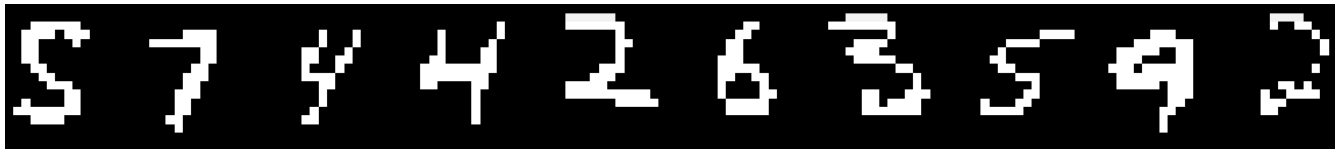


- Encoding is done in **two** dimensions. $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$.



Fun with MNIST – With simple model evaluation

- 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").

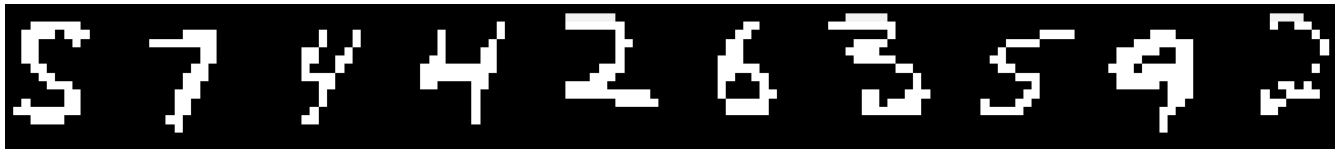


- Encoding is done in **two** dimensions. $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$.
- The **decoder network** $\mathbf{Z} \rightsquigarrow \mathbf{X}$ is a $64 + 256$ neural net with ReLU units.

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

Fun with MNIST – With simple model evaluation

- 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** $\mathbf{Z} \rightsquigarrow \mathbf{X}$.

Next up: Model validations – Dislaimer

The next few slides show **very simple** qualitative model critiques. These checks are by no means comprehensive, and in fact quite naïve.

See, e.g., D. Blei (2014): "*Build, Compute, Critique, Repeat: Data Analysis with Latent Variable Models*" and A. Gelman et al. (2020): "*Bayesian workflow*" for how it **should** be done.

Trying to reconstruct $\mathbf{x}$ by $\mathbb{E}_{p_\theta}[\mathbf{X} \mid \mathbf{Z} = \mathbb{E}_{q_\lambda}[\mathbf{Z} \mid \mathbf{x}]]$

An initial indication of performance:

1. For some $\mathbf{x}_0$, calculate $\mathbf{z}_0 \leftarrow \mathbb{E}_{q_\lambda}[\mathbf{Z} \mid \mathbf{X} = \mathbf{x}_0]$
2. ... and $\tilde{\mathbf{x}} \leftarrow \mathbb{E}_{p_\theta}[\mathbf{X} \mid \mathbf{Z} = \mathbf{z}_0]$.
3. Compare $\mathbf{x}_0$ and $\tilde{\mathbf{x}}$ visually.

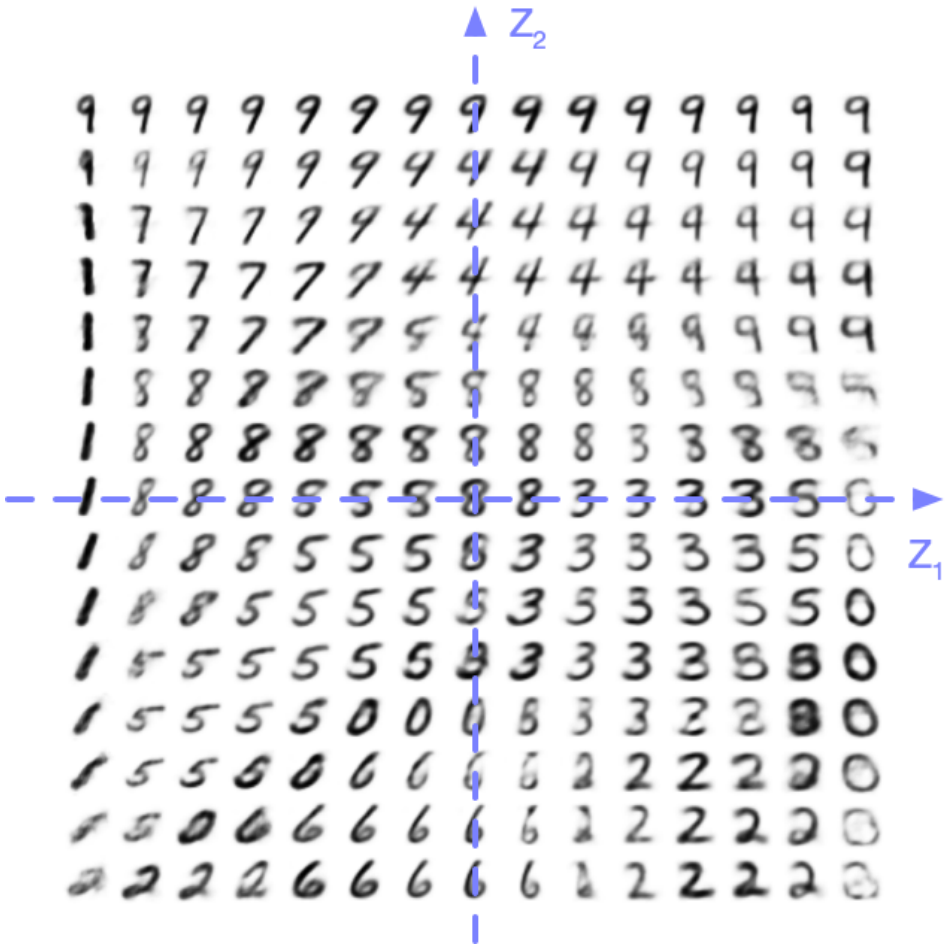
Test-set examples

Training examples (at end of training)

The picture manifold – $\mathbb{E}_{p_{\boldsymbol{\omega}}}[\mathbf{X} \mid \mathbf{z}]$ for different values of $\mathbf{z}$

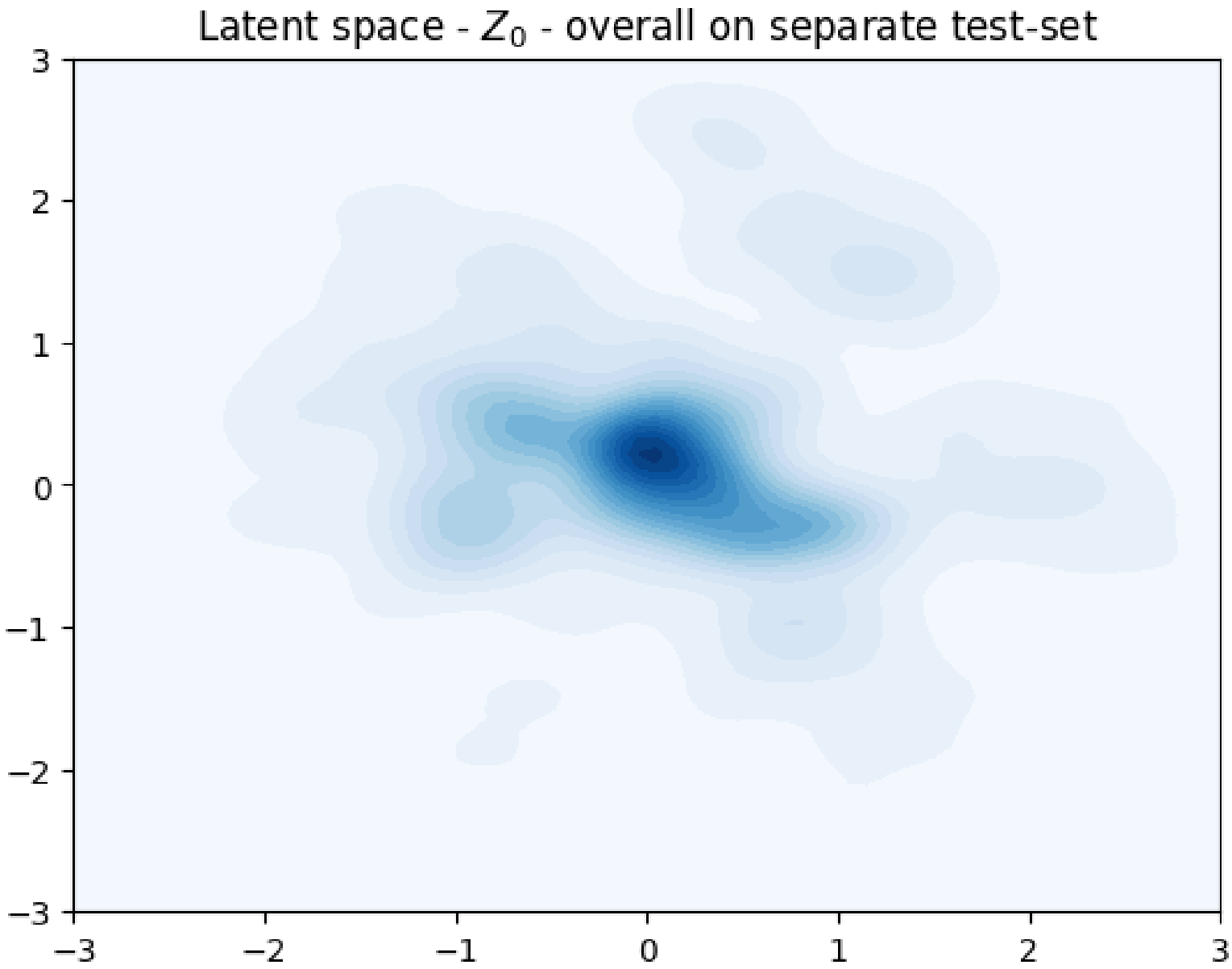
Using a VAE for generation

- The VAE is a **deep generative model** – albeit not a fancy one.
- **Process:** Sample $\mathbf{Z}_0 \sim p(\mathbf{z})$, then sample an $\mathbf{X} \sim p_{\boldsymbol{\omega}}(\mathbf{x} \mid \mathbf{z}_0)$.

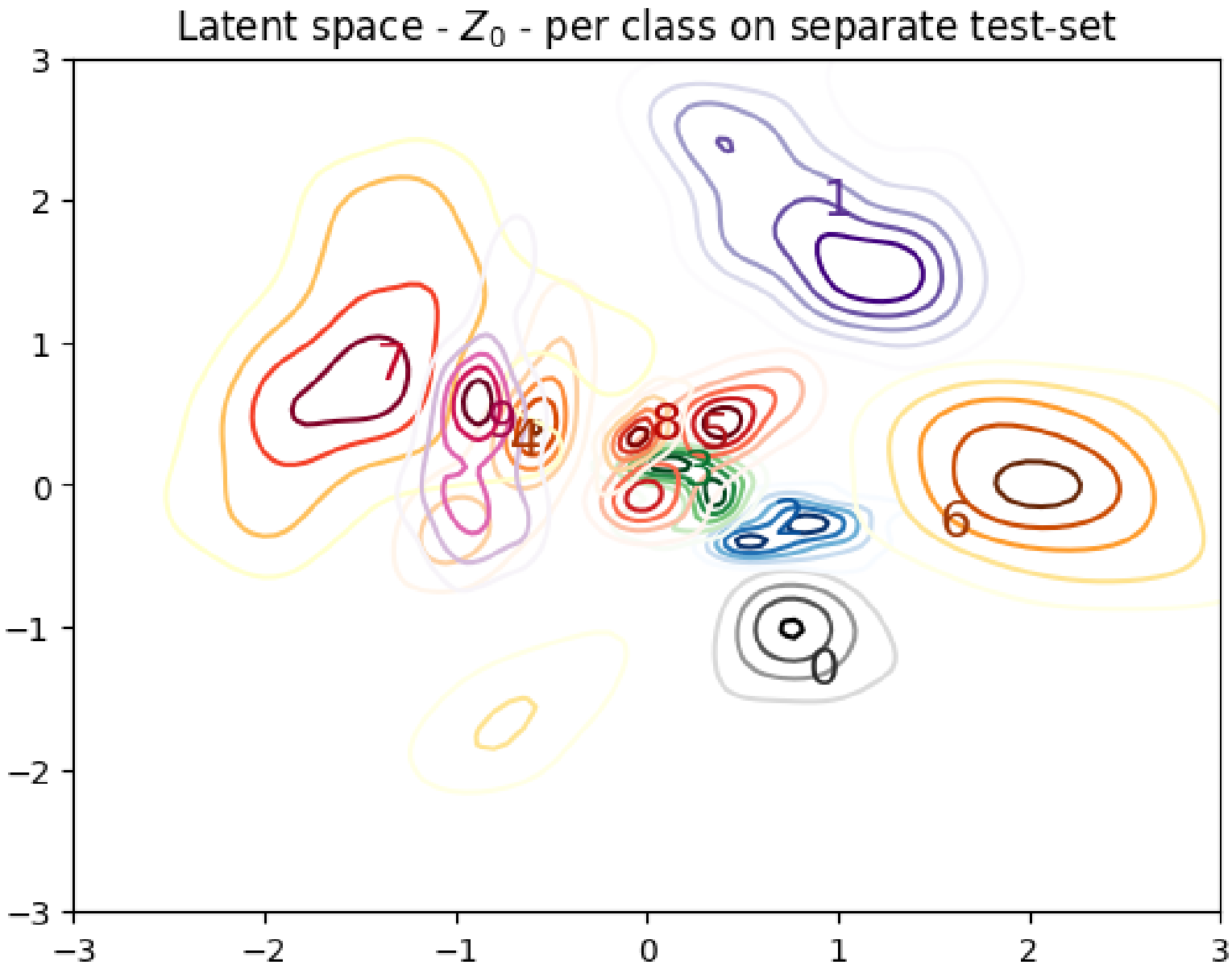


Generative ability, shown through $\mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\omega}}}[\mathbf{X} \mid \mathbf{z}]$ for different values of $\mathbf{z}$.

Averaged distribution over $\mathbb{Z}$



Averaged distribution over $\mathbb{Z}$ – per class



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

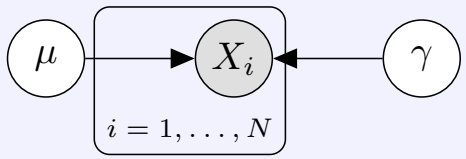
```
#The observatons
obs = {'sensor': torch.tensor(18.0)}

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`

Code Task: Play with VAEs

- We provide a VAE with a **linear decoder** implemented in Pyro. You will extend it!
- **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/students_VAE.ipynb`.

Conclusions

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

Recap & Conclusions

- **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, 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