

# Tropical Probabilistic AI school

## Variational Inference and Optimization

Helge Langseth, Bjørn Magnus Mathiesen, 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

## Why do we talk about this?

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

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

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

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

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

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

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

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

- 1 We may have to 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  $\nabla_{\lambda} \mathcal{L}(q(\theta | \lambda))$  exactly for (at least) two reasons:

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

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

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

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

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

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

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

# Black Box Variational Inference

## Main idea: Cast inference as an optimization problem

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

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

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

## Important issue:

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



The algorithm requires that we can find

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

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

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

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

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

The algorithm requires that we can find

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

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

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


Now it follows that

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

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

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

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

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


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

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

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

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

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

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

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

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

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

## Calculating the gradient – Things to notice

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

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

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

## Calculating the gradient – in summary

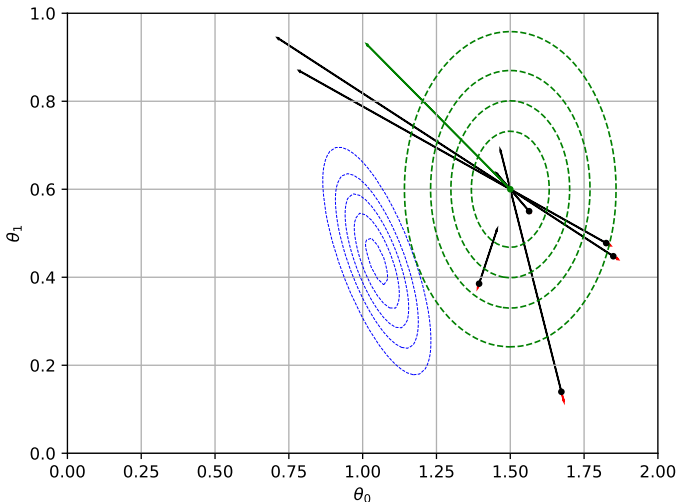
We have observed the data  $\mathcal{D}$ , and our current estimate for  $\boldsymbol{\lambda}$  is  $\hat{\boldsymbol{\lambda}}$ . Then

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

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



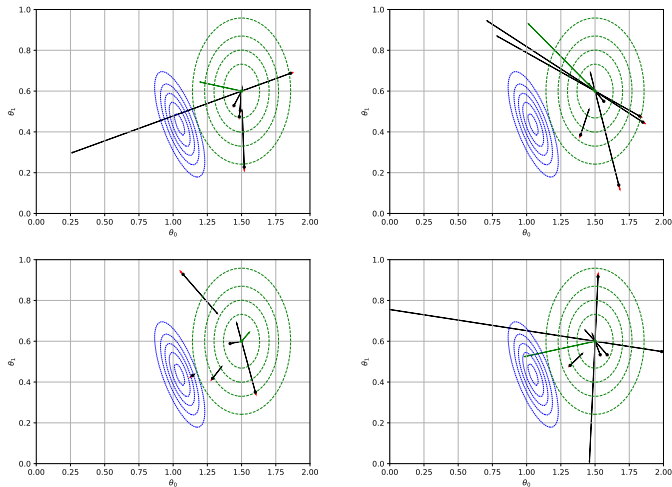
# Does it work?



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

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

# Does it work?

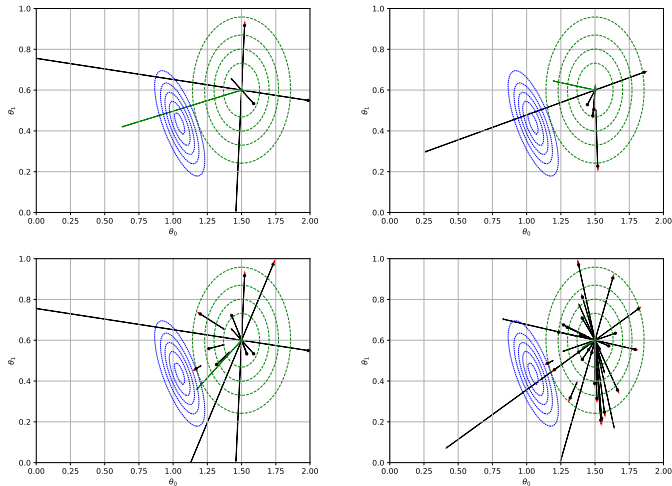


Different samples, each with  $M = 5$ .

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

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

# Does it work?



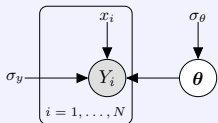
Different values of  $M$  ( $M = 3, 5, 10$ , and  $25$ )

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

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

# Does it work?

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



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

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

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

- Look at `Exercise 1` in the notebook

`Day2/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`.

Goal: Find a more robust estimator for the gradient

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

Goal: Find a more robust estimator for the gradient

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

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

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

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

Goal: Find a more robust estimator for the gradient

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

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

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

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

Example: The Gaussian distribution

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

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

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



**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:**

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

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

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

**Consequences for how we can calculate the gradient:**

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

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

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

**Consequences for how we can calculate the gradient:**

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

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

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

**Consequences for how we can calculate the gradient:**

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

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

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

**Consequences for how we can calculate the gradient:**

$$\begin{aligned}\nabla_{\lambda} \mathcal{L}(q) &= \nabla_{\lambda} \mathbb{E}_{\theta \sim q(\cdot|\lambda)} \left[ \log \frac{p(\theta, \mathcal{D})}{q(\theta|\lambda)} \right] \\ &= \nabla_{\lambda} \mathbb{E}_{\epsilon \sim \phi(\cdot)} \left[ \log \frac{p(f(\epsilon, \lambda), \mathcal{D})}{q(f(\epsilon, \lambda)|\lambda)} \right] \\ &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\lambda} \log \frac{p(f(\epsilon, \lambda), \mathcal{D})}{q(f(\epsilon, \lambda)|\lambda)} \right] \\ &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta|\lambda)} \nabla_{\lambda} f(\epsilon, \lambda) - \nabla_{\lambda} \log q(\theta|\lambda) \right] \\ &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta|\lambda)} \nabla_{\lambda} f(\epsilon, \lambda) \right] - 0 \quad (\text{See Slide 6, Point 3})\end{aligned}$$

## Monte-Carlo Estimation:

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

## Monte-Carlo Estimation:

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

## Monte-Carlo Estimation:

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



## Monte-Carlo Estimation:

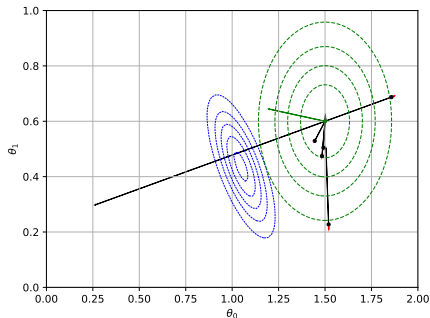
$$\begin{aligned}
 \nabla_{\lambda} \mathcal{L}(q) &= \mathbb{E}_{\epsilon \sim \phi} \left[ \nabla_{\theta} \log \frac{p(\theta, \mathcal{D})}{q(\theta | \lambda)} \nabla_{\lambda} f(\epsilon, \lambda) \right] \\
 &\approx \frac{1}{M} \sum_{j=1}^M \left[ \nabla_{\theta} \log \frac{p(\theta_j, \mathcal{D})}{q(\theta_j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

## This gradient estimator...

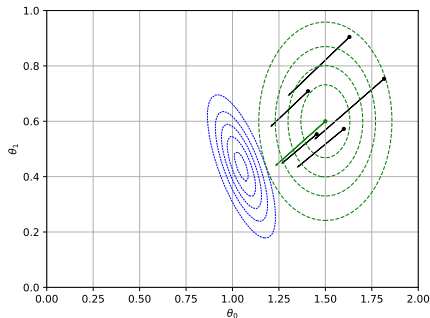
- Uses the *model's* gradients (not so for the score-function gradient).
- Requires  $q(\theta | \lambda)$  to be *reparametrizable* and *differentiable*.
- Requires  $\log p(\theta, \mathcal{D})$  to be differentiable wrt.  $\theta$ .

# Does it work?

Score-function gradient



Reparameterized gradient



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

Notice the direction of each sample's gradient:

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

## Score function gradients:

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

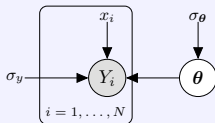
## Reparametrization gradients:

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

## Conclusion

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

## Code Task: Reparameterization-gradient for linear regression



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

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

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

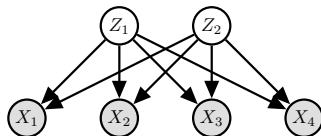
- We provide  $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}, \mathcal{D})$ ,  $\nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$  and  $\nabla_{\boldsymbol{\lambda}} f(\boldsymbol{\epsilon}, \boldsymbol{\lambda})$  for this model.
- Go to Exercise 2 in

`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

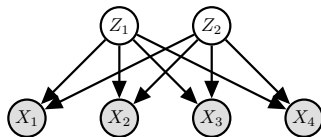


$$\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

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

- The FA model posits that the data  $\mathbf{X}$  can be generated from **independent factors**  $\mathbf{Z}$  pluss some sensor-noise:  $\mathbf{X} | \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^T \mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ .
- **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



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

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

- The FA model posits that the data  $\mathbf{X}$  can be generated from **independent factors**  $\mathbf{Z}$  pluss some sensor-noise:  $\mathbf{X} | \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^T \mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ .
- **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.

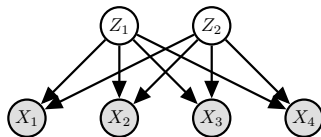
## Example: Grades

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

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

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

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



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

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

- The FA model posits that the data  $\mathbf{X}$  can be generated from **independent factors**  $\mathbf{Z}$  plus some sensor-noise:  $\mathbf{X} | \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^T \mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ .
- **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.

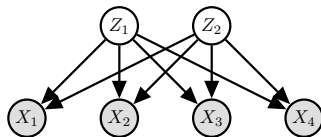
How do we feel about the FA model?

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

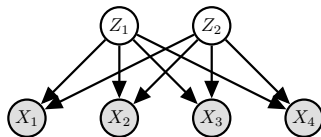
VAE:  $\mathbf{X} | \mathbf{z} \sim$  "Whatever"

- The FA model posits that the data  $\mathbf{X}$  can be generated from **independent factors**  $\mathbf{Z}$  plus some sensor-noise:  $\mathbf{X} | \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^T \mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ .
- **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.

## From Factor Analysis to Variational Auto Encoders

VAEs allow the distribution  $p(\mathbf{x} | \mathbf{z})$  to be **arbitrarily complex** – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate  $p(\mathbf{z} | \mathbf{x})$ , 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:  $\mathbf{X} | \mathbf{z} \sim$  “Whatever”

- The FA model posits that the data  $\mathbf{X}$  can be generated from **independent factors**  $\mathbf{Z}$  plus some sensor-noise:  $\mathbf{X} | \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^T \mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ .
- **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.

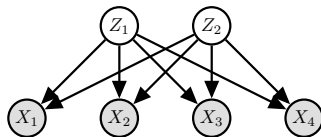
## From Factor Analysis to Variational Auto Encoders

VAEs allow the distribution  $p(\mathbf{x} | \mathbf{z})$  to be **arbitrarily complex** – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate  $p(\mathbf{z} | \mathbf{x})$ , 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  $\mathbf{x} \xrightarrow{p(\mathbf{z} | \mathbf{x})} \mathbf{z} \xrightarrow{p(\mathbf{x} | \mathbf{z})} \hat{\mathbf{x}}$  (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:  $\mathbf{X} | \mathbf{z} \sim$  “Whatever”

- The FA model posits that the data  $\mathbf{X}$  can be generated from **independent factors**  $\mathbf{Z}$  pluss some sensor-noise:  $\mathbf{X} | \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^T \mathbf{z} + \boldsymbol{\epsilon}; \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ .
- **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.

## From Factor Analysis to Variational Auto Encoders

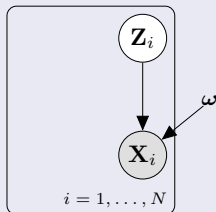
VAEs allow the distribution  $p(\mathbf{x} | \mathbf{z})$  to be **arbitrarily complex** – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate  $p(\mathbf{z} | \mathbf{x})$ , and it is therefore harder to interpret the factors  $\mathbf{Z}$ .

### Why that name?

VAEs are called **auto-encoders** because we can train them by “re-creating” inputs via the process  $\mathbf{x} \xrightarrow{p(\mathbf{z} | \mathbf{x})} \mathbf{z} \xrightarrow{p(\mathbf{x} | \mathbf{z})} \hat{\mathbf{x}}$  (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

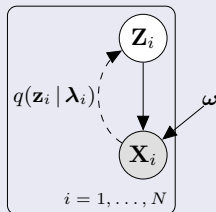


- $p(\mathbf{z}_i)$  is (usually) an isotropic Gaussian distribution.
- $p_{\omega}(\mathbf{x}_i | g_{\omega}(\mathbf{z}_i))$ , where  $g$  is a deep neural network.

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

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

## Model of interest



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

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

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

## Variational Inference

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

$$p_{\omega}(\mathbf{z}_i | \mathbf{x}_i) = \frac{p_{\omega}(\mathbf{z}_i) \cdot p_{\omega}(\mathbf{x}_i | g_{\omega}(\mathbf{z}_i))}{\int_{\mathbf{z}_i} p_{\omega}(\mathbf{z}_i) \cdot p_{\omega}(\mathbf{x}_i | g_{\omega}(\mathbf{z}_i)) d\mathbf{z}_i}.$$

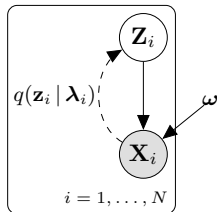
- **Initial plan:** Fit  $q(\mathbf{z}_i | \lambda_i)$  to  $p_{\omega}(\mathbf{z}_i | \mathbf{x}_i)$  using variational inference.

## Initial plan:

- Optimize the ELBO

$$\mathcal{L}(\omega, \lambda_1, \dots, \lambda_N) = -\mathbb{E}_q \left[ \log \frac{\prod_{i=1}^N q(\mathbf{z}_i | \lambda_i)}{\prod_{i=1}^N p_{\omega}(\mathbf{z}_i, \mathbf{x}_i)} \right].$$

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

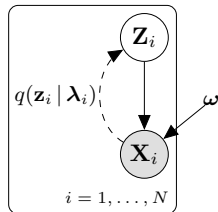


## Initial plan:

- Optimize the ELBO

$$\mathcal{L}(\omega, \lambda_1, \dots, \lambda_N) = -\mathbb{E}_q \left[ \log \frac{\prod_{i=1}^N q(\mathbf{z}_i | \lambda_i)}{\prod_{i=1}^N p_{\omega}(\mathbf{z}_i, \mathbf{x}_i)} \right].$$

- A natural model for  $q(\mathbf{z}_i | \lambda_i)$  is a Gaussian with parameters  $\lambda_i = \{\mu_i, \Sigma_i\}$ .
- If  $\mathbf{Z}_i$  is  $d$ -dim and we for simplicity assume diagonal  $\Sigma_i$ , 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  $g_{\omega}(\mathbf{z})$  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  $\lambda_i$  and  $\lambda_j$  should be “close”, too.
- **Therefore:** Let’s assume there exists a (smooth) function  $h(\mathbf{x})$  so that  $h(\mathbf{x}_i) = \lambda_i$ .
- $h(\cdot)$  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!

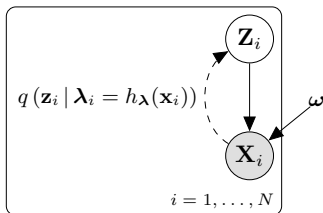
## Benefits:

- 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:

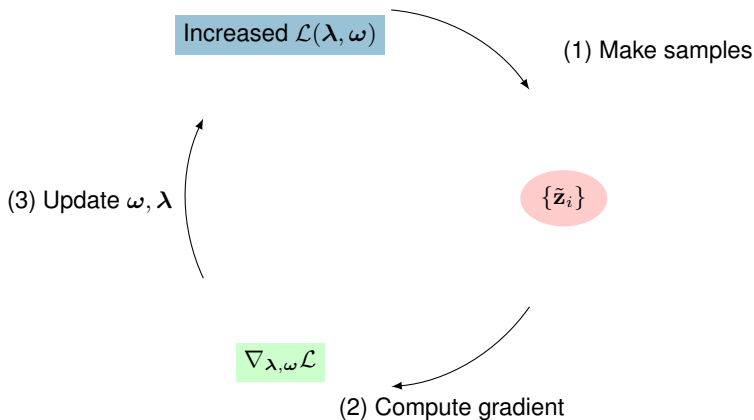
- $p(\mathbf{z}_i)$  is an isotropic Gaussian distribution.
- $p_{\omega}(\mathbf{x}_i | \mathbf{z}_i) \sim \text{Bernoulli}(\text{logits} = g_{\omega}(\mathbf{z}_i))$ ,  
where  $g_{\omega}$  is a DNN with weights  $\omega$ .
- $q(\mathbf{z}_i | \mathbf{x}_i, \lambda) \sim \mathcal{N}(\mu_i, \Sigma_i)$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



## Goal:

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

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



- 1 For each  $\mathbf{x}_i$ , sample  $M$  (typically  $M = 1$ )  $\epsilon$ -values to get  $\mathbf{z}_i$  samples.
- 2 Calculate  $\nabla_{\lambda, \omega} \mathcal{L}(\lambda, \omega)$  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  $p(\mathcal{D}|\theta)$  and the prior  $p(\theta)$ .
- 2 **Automatic** : Define the set of variational distributions  $q(\theta|\lambda) \in \mathcal{Q}$ .  
(In *complicated* situations this step may have to be **Manual** ).
- 3 **Automatic** : Optimize ELBO:  $\lambda_{t+1} = \lambda_t + \rho \nabla_{\lambda} \mathcal{L}(\lambda_t)$  using an AutoDiff. engine.
- 4 **Automatic** : Find  $q(\theta|\lambda^*) = \arg \min_{q \in \mathcal{Q}} \text{KL} (q(\theta|\lambda) || p(\theta|\mathcal{D}))$ .

## Probabilistic Programming Languages and Box's loop

Modern PPLs relieve us of all the computational details!

Instead we can focus on . . .

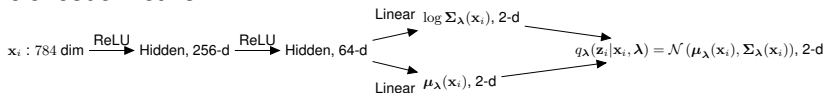
- Building models (define  $p(\mathcal{D}|\theta)$  and  $p(\theta)$ ) we believe in.
- Using computed results to validate/critique and iteratively refine the model.

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

- 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  $28 \times 28$  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 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  $28 \times 28$  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.

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

- 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  $28 \times 28$  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**  $\mathbf{Z} \rightsquigarrow \mathbf{X}$ .

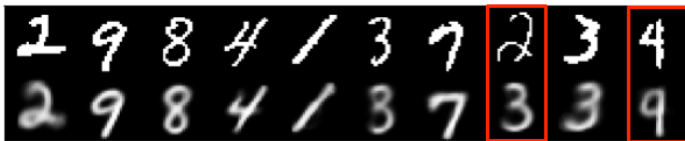
## Next up: Model validations – Disclaimer

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.

An initial indication of performance:

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



Test-set examples

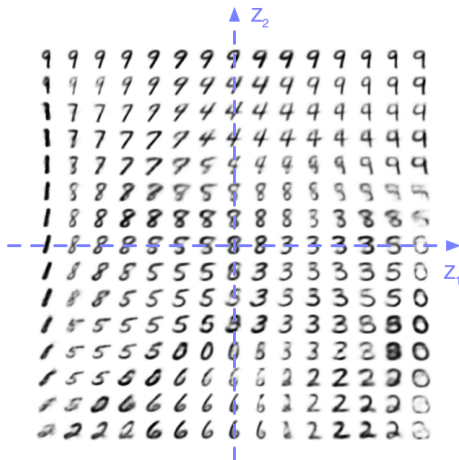


Training examples (at end of training)

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

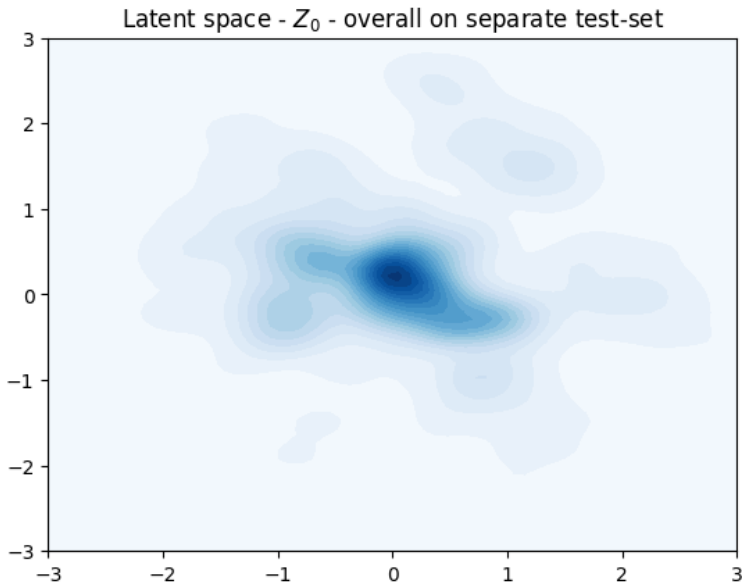
## Using a VAE for generation

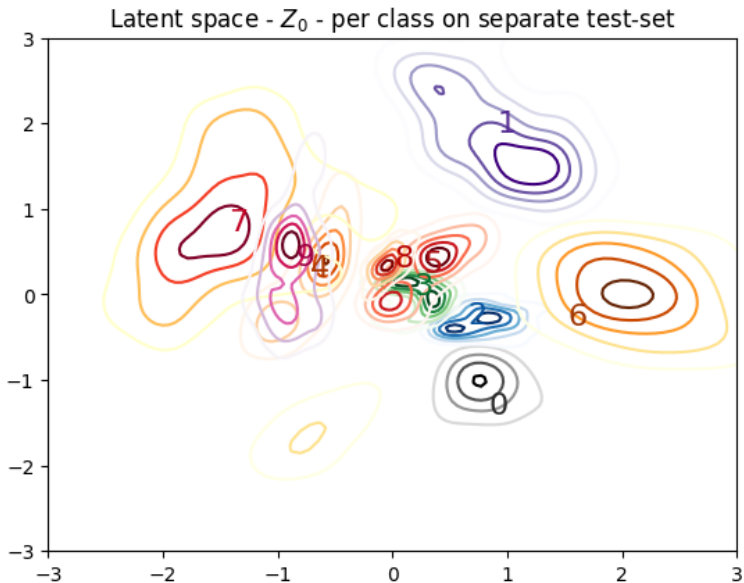
- The VAE is a **deep generative model** – albeit not a fancy one.
- **Process:** Sample  $\mathbf{Z}_0 \sim p(\mathbf{z})$ , then sample an  $\mathbf{X} \sim p_{\omega}(\mathbf{x} | \mathbf{z}_0)$ .



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







I have just imported in the file AL-Part-5.tex here.

- It contains too much, so cut down dramatically and make some comments on Pyro so the students can relate.
- In particular, one of the first coding task about guides must go!
- Most work to be done by updating the notebook, maybe? Comments there to say what goes on etc

# Probabilistic programming: Variational inference in Pyro

## Pyro

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

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

... and there are also many other possibilities

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

## Simple example

$$\begin{aligned}\text{temp} &\sim \mathcal{N}(15, 2) \\ \text{sensor} &\sim \mathcal{N}(\text{temp}, 1) \\ p(\text{sensor} = 18, \text{temp})\end{aligned}$$

## Simple example

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

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

## 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

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



## Inference Problem

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

## Inference Problem

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

## Variational Solution

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

## Inference Problem

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

## Variational Solution

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

## Pyro Guides:

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

## Inference Problem

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

## Variational Solution

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

## Pyro Guides:

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

## Pyro Guides:

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

## Pyro Guides:

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

## Guide requirements

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

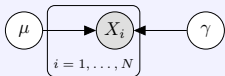
## Example

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

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

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

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



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

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

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

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

Day2-BeforeLunch/student\_simple\_model.ipynb

Clean up previous stuff. Mostly not needed.



## 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  $p(\theta \mid \mathcal{D})$

- **Bayesian Machine Learning**

- **Variational inference**

- **Provides**  $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$ : A distributional approximation to  $p(\boldsymbol{\theta} \mid \mathcal{D})$
- **Objective:**  $\arg \min_{\boldsymbol{\lambda}} \text{KL} (q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) \parallel p(\boldsymbol{\theta} \mid \mathcal{D})) \Leftrightarrow \arg \max_{\boldsymbol{\lambda}} \mathcal{L} (q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}))$
- **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*)

- **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!**

- **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, ...)

- **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