

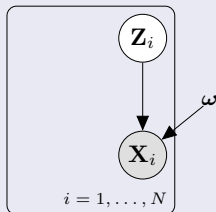
Nordic probabilistic AI school  
Variational Inference and Optimization

Helge Langseth, Andrés Masegosa, and Thomas Dyhre Nielsen

June 13, 2023

# Deep Bayesian Learning – VAE

## Model of interest

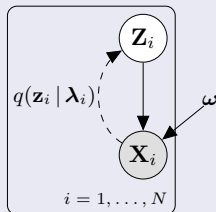


- $p(\mathbf{z}_i)$  is (usually) an isotropic Gaussian distribution.
- $p_{\omega}(\mathbf{x}_i | g_{\omega}(\mathbf{z}_i))$ , where  $g$  is a deep neural network.

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

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

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

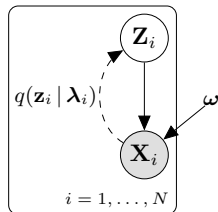
- **Initial plan:** Fit  $q(\mathbf{z}_i | \lambda_i)$  to  $p_{\omega}(\mathbf{z}_i | \mathbf{x}_i)$  using variational inference.

## 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.

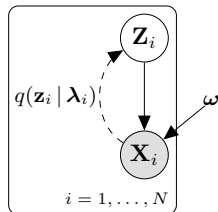


## Initial plan:

- 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.



## A better plan

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

$\rightsquigarrow \lambda_i$  and  $\lambda_j$  should be “close” if  $\mathbf{x}_i$  and  $\mathbf{x}_j$  are “close”.

- **Therefore:** Let’s assume there exists a (smooth) function  $h(\mathbf{x})$  so that  $h(\mathbf{x}_i) = \lambda_i$ .
- $h(\cdot)$  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  $\boldsymbol{\lambda}$ .

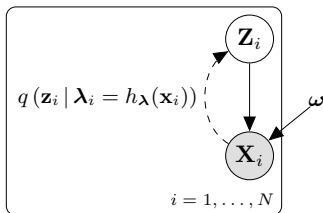
**Note!** Amortized inference is useful also outside VAEs!

## 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  $\lambda$ .

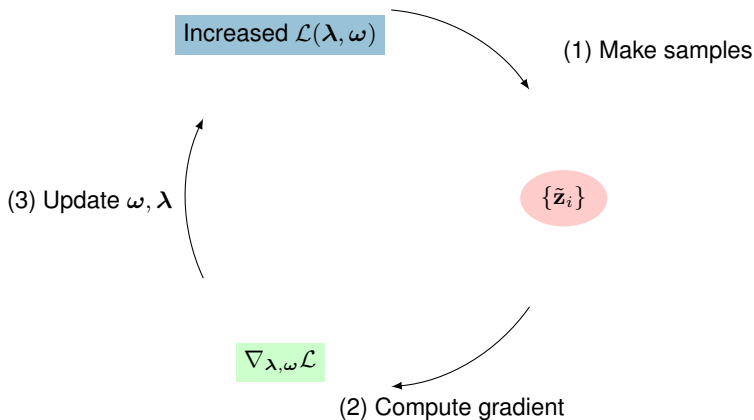


## Goal:

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 1)  $\epsilon$ -values.
- ② Calculate  $\nabla_{\lambda, \omega} \mathcal{L}(\lambda, \omega)$  using the reparameterization-trick.
- ③ Update parameters using a standard DL optimizer (like Adam).

- 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”).

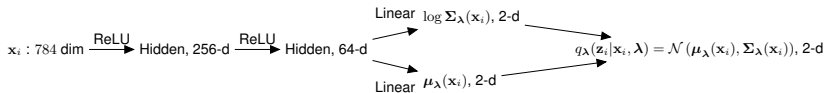


- The model is learned from  $N = 55.000$  training examples.
- Each  $\mathbf{x}_i$  is a binary vector of 784 pixel values.
- When seen as a  $28 \times 28$  array, each  $\mathbf{x}_i$  is a picture of a handwritten digit (“0” – “9”).



- 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.

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

## Code Task: VAEs in Pyro

- Learn how a VAE is coded in Pyro.
- We provide a VAE with a **linear decoder**.
- **Exercise 1: Define a Non-Linear Decoder**
  - A MLP with a hidden layer with non-linearities (e.g. Relu).
- **Exercise 2: Explore the latent space**
  - Moving from linear to non-linear decoders with different capacity.
- Notebook:

`Day2-Evening/students_VAE.ipynb`.

# Conclusions

- **Bayesian Machine Learning**

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

- **Bayesian Machine Learning**

- **Variational inference**

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