

Nordic probabilistic AI school

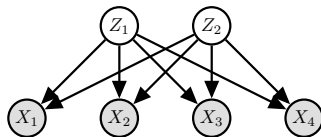
Variational Inference and Optimization

Helge Langseth, Andrés Masegosa, and Thomas Dyhre Nielsen

June 17, 2025

Deep Bayesian Learning – VAE

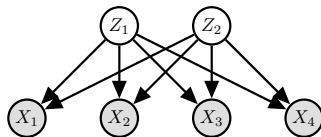
Starting-point: The factor analysis model, and an extension



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

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

- 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**.



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

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

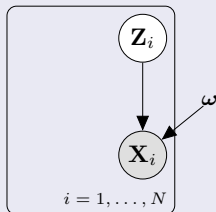
- The FA model posits that the data \mathbf{X} can be generated from **independent factors** \mathbf{Z} pluss some sensor-noise: $\mathbf{X} | \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^T \mathbf{z} + \boldsymbol{\epsilon}$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
-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

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.

Model of interest

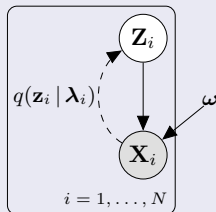


- $p(\mathbf{z}_i)$ is (usually) an isotropic Gaussian distribution.
- $p_{\omega}(\mathbf{x}_i | g_{\omega}(\mathbf{z}_i))$, where g is a deep neural network.

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

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

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 ω to maximize the model's fit to \mathcal{D} .
 - We will cheat and find a **point estimate** for ω .

Variational Inference

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

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

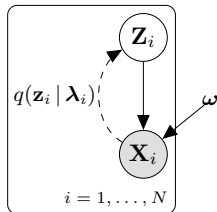
- **Initial plan:** Fit $q(\mathbf{z}_i | \lambda_i)$ to $p_{\omega}(\mathbf{z}_i | \mathbf{x}_i)$ using variational inference.

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 Σ_i , this still gives **$2Nd$ variational parameters** to learn.
- An $\tilde{\mathbf{x}} \notin \mathcal{D}$ at query time will be **problematic**.

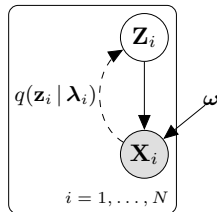


Initial plan:

- 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 Σ_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”: if \mathbf{z}_i and \mathbf{z}_j are “close”, then so are \mathbf{x}_i and \mathbf{x}_j .

$\rightsquigarrow \lambda_i$ and λ_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 $\boldsymbol{\lambda}$.

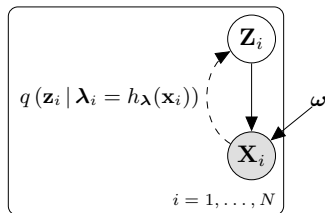
Note! Amortized inference is useful also outside VAEs!

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:

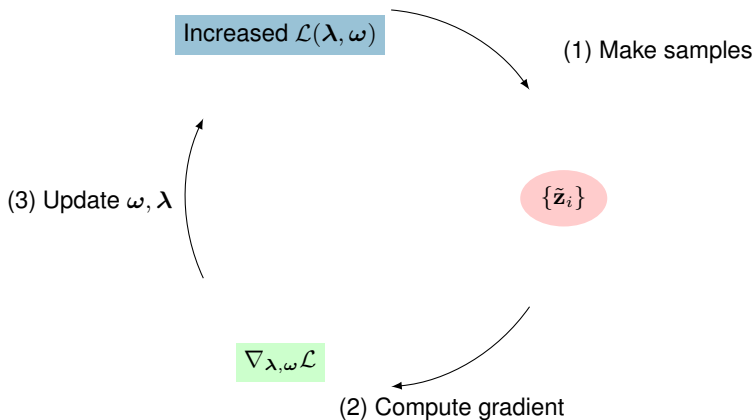
- $p(\mathbf{z}_i)$ is an isotropic Gaussian distribution.
- $p_{\omega}(\mathbf{x}_i | \mathbf{z}_i) \sim \text{Bernoulli}(\text{logits} = g_{\omega}(\mathbf{z}_i))$,
where g_{ω} is a DNN with weights ω .
- $q(\mathbf{z}_i | \mathbf{x}_i, \lambda) \sim \mathcal{N}(\mu_i, \Sigma_i)$,
where $\{\mu_i,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



Goal:

Learn **both** ω and λ 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) ϵ -values.
- 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 . . .

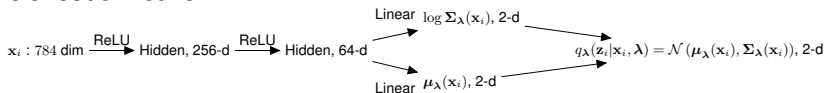
- Building models (define $p(\mathcal{D}|\theta)$ and $p(\theta)$) we believe in.
- Using computed results to validate/critique and iteratively refine the model.

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 28×28 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 28×28 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 28×28 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** $\mathbf{Z} \rightsquigarrow \mathbf{X}$.

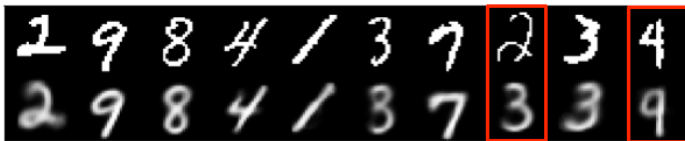
Next up: Model validations – Disclaimer

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

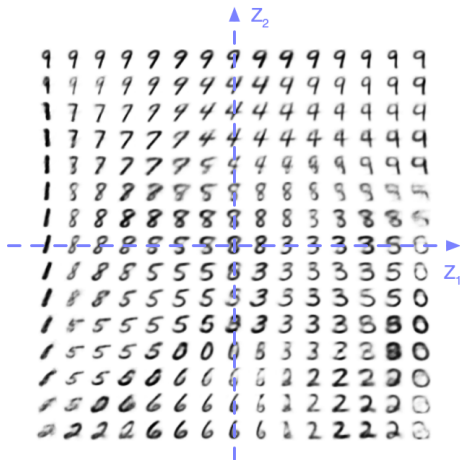


Training examples (at end of training)

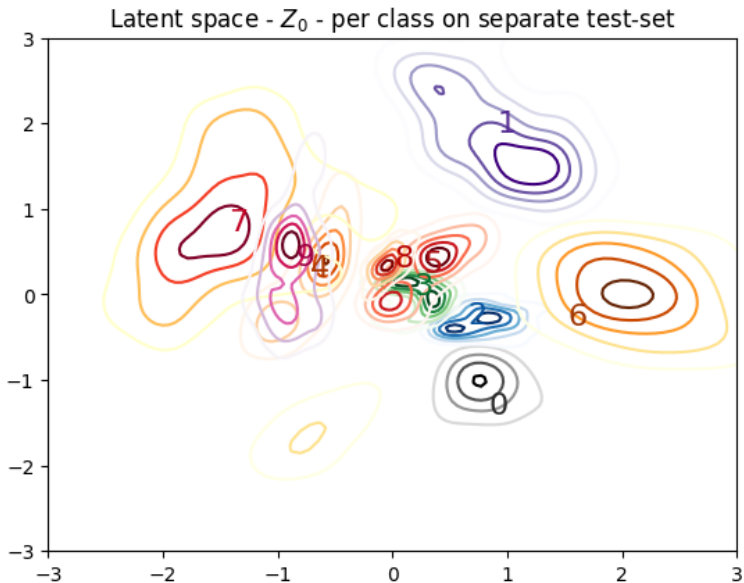
The picture manifold – $\mathbb{E}_{p_{\omega}}[\mathbf{X} | \mathbf{z}]$ for different values of \mathbf{z}

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} .



Code Task: VAEs in Pyro

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

`Evening/students_VAE.ipynb.`

Conclusions

- **Bayesian Machine Learning**

- Represents unobserved quantities using **distributions**
- Models **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, 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