Nordic probabilistic AI school

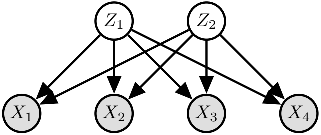
Variational Inference and Optimization

Helge Langseth, Andrés Masegosa, and Thomas Dyhre Nielsen

June 18, 2024

Deep Bayesian Learning – VAE

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

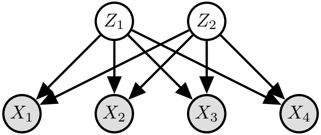


$$\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

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

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



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

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

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

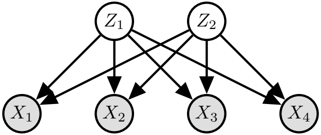
Example: Grades

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



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

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

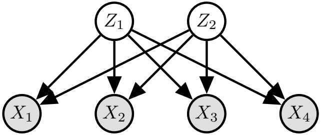
- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \mid \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^{\mathsf{T}}\mathbf{z} + \boldsymbol{\epsilon}$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.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

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



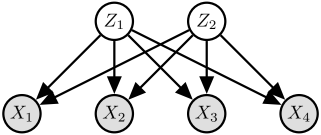
VAE: $\mathbf{Z} \sim$ "Whatever", typically still $\mathcal{N}(\mathbf{0}, \mathbf{I})$

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 $\mathbf{Z}$.



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: $\mathbf{X} \,|\, \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^{\mathsf{T}}\mathbf{z} + \boldsymbol{\epsilon}$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **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.

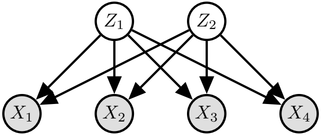
From Factor Analysis to Variational Auto Encoders

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 $\mathbf{x} \overset{p(\mathbf{z}\,|\,\mathbf{x})}{\rightsquigarrow} \mathbf{z} \overset{p(\mathbf{x}\,|\,\mathbf{z})}{\rightsquigarrow} \hat{\mathbf{x}}$ (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: $\mathbf{X} \,|\, \mathbf{z} \sim$ "Whatever"

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

From Factor Analysis to Variational Auto Encoders

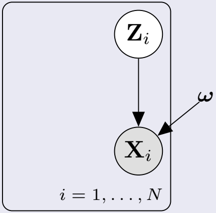
VAEs allow the distribution $p(\mathbf{x} \,|\, \mathbf{z})$ to be **arbitrarily complex** – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate $p(\mathbf{z} \,|\, \mathbf{x})$, and it is therefore harder to interpret the factors $\mathbf{Z}$.

Why that name?

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



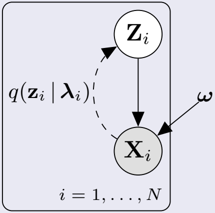
- $p(\mathbf{z}_i)$ is (usually) an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))$, where g is a deep neural network.

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

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

The Variational Auto Encoder (VAE)

Model of interest



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

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

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

Variational Inference

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

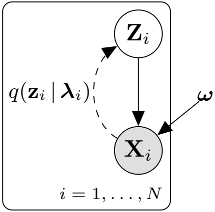
$$p_{\boldsymbol{\omega}}(\mathbf{z}_i \mid \mathbf{x}_i) = \frac{p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))}{\int_{\mathbf{z}_i} p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))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

- **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

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

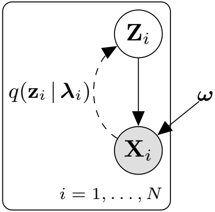


- A natural model for $q(\mathbf{z}_i \,|\, \boldsymbol{\lambda}_i)$ is a Gaussian with parameters $\boldsymbol{\lambda}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$.
- If $\mathbf{Z}_i$ is d-dim and we for simplicity assume diagonal $\boldsymbol{\Sigma}_i$, this still gives $2Nd$ variational parameters 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": if $\mathbf{z}_i$ and $\mathbf{z}_j$ are "close", then so are $\mathbf{x}_i$ and $\mathbf{x}_j$.

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

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

Amortized inference:

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

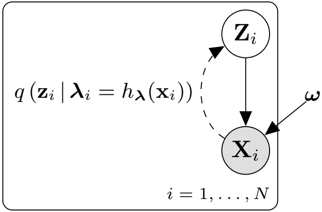
Note! Amortized inference is useful also outside VAEs!

Benefits:

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

The full VAE approach:

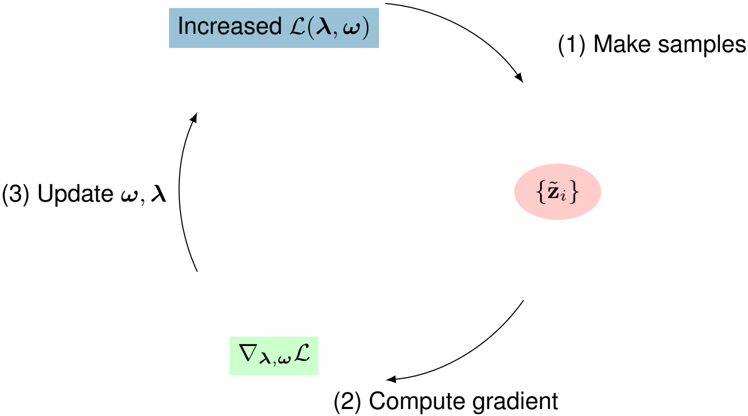
- $p(\mathbf{z}_i)$ is an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i|\mathbf{z}_i) \sim$ Bernoulli(logits $= g_{\boldsymbol{\omega}}(\mathbf{z}_i)$), where $g_{\boldsymbol{\omega}}$ is a DNN with weights $\boldsymbol{\omega}$.
- $q(\mathbf{z}_i \,|\, \mathbf{x}_i, \boldsymbol{\lambda})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



Goal:

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

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



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

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

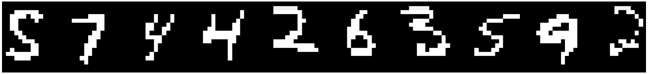
Modern PPLs relieve us of all the computational details!

Instead we can focus on . . .

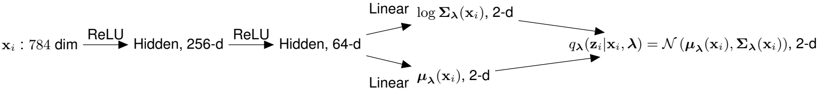
- Building models (define $p(\mathcal{D}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$) we believe in.
- Using computed results to validate/critique and iteratively refine the model.

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

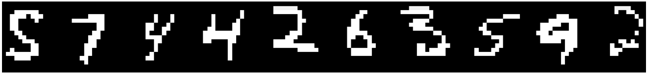
- The model is learned from $N = 55.000$ training examples.
- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit ("0" – "9").



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



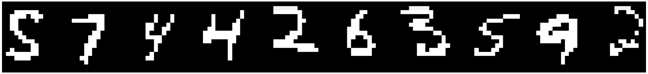
- The model is learned from $N = 55.000$ training examples.
- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit (“0” – “9”).



- Encoding is done in **two** dimensions. $p(\mathbf{z}_i) = \mathcal{N}\left(\mathbf{0}_2, \mathbf{I}_2\right)$.
- 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 dim $\xrightarrow{\text{ReLU}}$ Hidden, 64-d $\xrightarrow{\text{ReLU}}$ Hidden, 256-d $\xrightarrow{\text{Linear}}$ logit$(\mathbf{p}_i)$, 784-d $\longrightarrow$ $p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid \mathbf{z}_i, \boldsymbol{\omega}) =$ Bernoulli$\left(\mathbf{p}_i\right)$, 784-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 – 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:

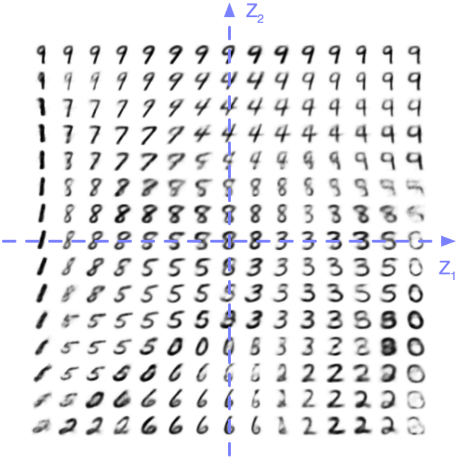
1. For some $\mathbf{x}_0$, calculate $\mathbf{z}_0 \leftarrow \mathbb{E}_{q_\lambda}[\mathbf{Z} \mid \mathbf{X} = \mathbf{x}_0]$
2. ... and $\tilde{\mathbf{x}} \leftarrow \mathbb{E}_{p_\theta}[\mathbf{X} \mid \mathbf{Z} = \mathbf{z}_0]$.
3. Compare $\mathbf{x}_0$ and $\tilde{\mathbf{x}}$ visually.

Test-set examples

Training examples (at end of training)

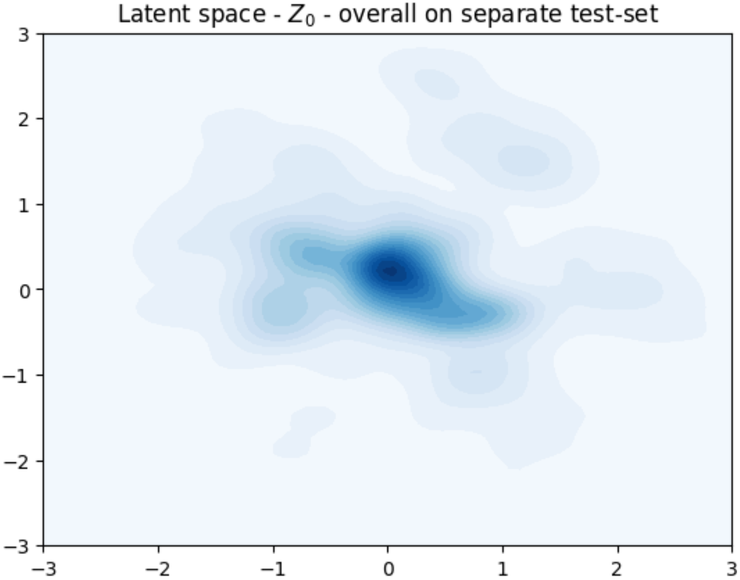
Using a VAE for generation

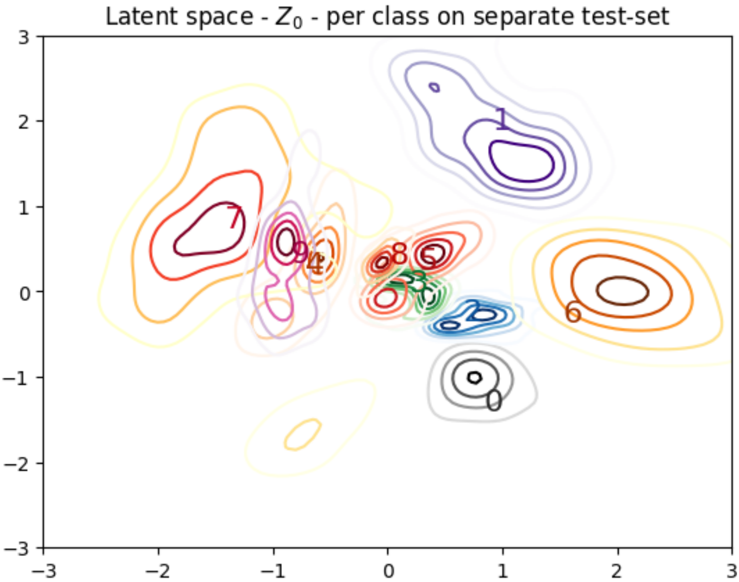
- The VAE is a **deep generative model** – albeit not a fancy one.
- **Process:** Sample $\mathbf{Z}_0 \sim p(\mathbf{z})$, then sample an $\mathbf{X} \sim p_{\boldsymbol{\omega}}(\mathbf{x} \mid \mathbf{z}_0)$.



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 $\mathbf{Z}$




Latent space - Z_0 - per class on separate test-set
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
1
7
0
9
4
8
5
3
6
0

Code Task: VAEs in Pyro

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

  `Day2-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:** $\arg\min_{\boldsymbol{\lambda}} \mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) || p(\boldsymbol{\theta} \mid \mathcal{D})\right) \Leftrightarrow \arg\max_{\boldsymbol{\lambda}} \mathcal{L}\left(q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})\right)$
  - **Mean-field:** Divide and conquer strategy for high-dimensional posteriors
  - **Main caveat:** $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$ underestimates the uncertainty of $p(\boldsymbol{\theta} \mid \mathcal{D})$

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

Recap & Conclusions

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

Recap & Conclusions

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

- **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