Probabilistic AI

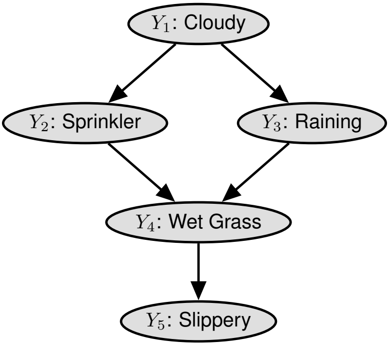
Lecture 2: Disentanglement in the variational auto encoder

Helge Langseth*

Jan. 2021

* Kudos to Anna Rodum Bjøru, NTNU. Slides made together with Thomas D. Nielsen, Aalborg University.

Summary from yesterday



- Each node is a random variable
- Edges indicate "influence" (Math-def: Graph encodes cond.indep. statements)
- For each variable Y_k, we must define $p(y_k \mid \mathrm{pa}(y_k))$.
- The full model is defined as $p(\mathbf{y}) = p(y_1, \ldots, y_n) = \prod_{i=1}^{n} p(y_i \mid \mathrm{pa}(y_i))$.
- Markov properties $\Leftrightarrow$ Factorisation property.

Inference: Calculating $p(\mathbf{z} \mid \mathbf{x})$

Our focus yesterday was on **approximate Inference**:
How to efficiently approximate $p(\mathbf{z} \mid \mathbf{x})$ by a simpler $q(\mathbf{z} \mid \mathbf{x})$.

- Looking for a “good” approximation means minimizing $\mathrm{KL}\left(q(\mathbf{z}) || p(\mathbf{z} \mid \mathbf{x})\right)$.
  - The distance measure has weaknesses, in particular **zero-forcing** behaviour.
  - Instead of minimizing the KL, we reformulated to maximizing the ELBO.
- Our set of candidate functions is $\mathcal{Q} = \{$All distributions that factorize$\}$.
  - Each local distribution $q_i(z_i \mid \boldsymbol{\lambda}_i)$ needs some pre-selected distributional family.
  - ... while we get to play with the parameters $\boldsymbol{\lambda}_i$.
  - Be aware that the MF assumption can reinforce the zero-forcing behaviour.
- We decided to optimize the parameters using BBVI (stochastic gradient ascent).
  - BBVI has some issues on its own, that we did not cover.
- We are now ready to combine this with other “compatible” pieces of machine learning.

Plan for my part of the winter-school

Day 1: Introduction to variational inference and the ELBO
Dive into the mathematical details of Probabilistic AI, understand the foundation, and investigate the effects of some of the "shortcuts" being made.

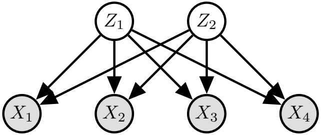
- **Approximate inference** via the KL divergence, a.k.a. **Variational Bayes**
- The **mean-field** approach to Variational Bayes
- **Black Box variational inference**

Day 2: Disentanglement in the variational auto encoder
Devise flexible models for representation learning, and consider their transparency.

- **Variational Auto Encoders**
- **Disentanglement:** What, why, how?
- **Probabilistic Programming Languages**

Variational Auto-Encoders

The factor analysis model, and an extension

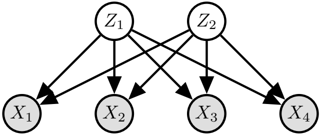


$$\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

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

- 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.

The factor analysis model, and an extension



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

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

- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \mid \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\intercal \mathbf{z} + \boldsymbol{\epsilon}$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \mid \mathbf{x})$ (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.

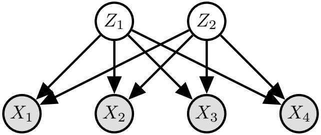
Example: Grades

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

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

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

The factor analysis model, and an extension



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

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

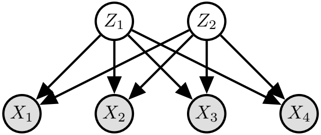
- The FA model posits that the data $\mathbf{X}$ can be generated from **independent factors** $\mathbf{Z}$ pluss some sensor-noise: $\mathbf{X} \,|\, \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\intercal \mathbf{z} + \boldsymbol{\epsilon}$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.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

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.

The factor analysis model, and an extension



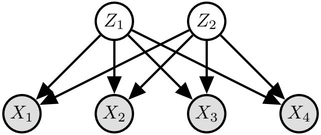
VAE: $\mathbf{Z} \sim$ "Whatever", typically still $\mathcal{N}(\mathbf{0}, \mathbf{I})$

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: $\mathbf{X} \mid \mathbf{z} = \boldsymbol{\mu} + \mathbf{W}^\intercal \mathbf{z} + \boldsymbol{\epsilon}$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$.
- **Simple algorithms** to find estimators $\hat{\boldsymbol{\mu}}$, $\hat{\mathbf{W}}$, and $\hat{\boldsymbol{\Sigma}}$, and closed form expression for $p(\mathbf{z} \mid \mathbf{x})$ (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.

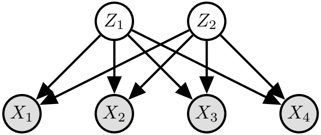
From Factor Analysis to Variational Auto Encoders

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 $\mathbf{x} \approx \hat{\mathbf{x}}$).

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.

Building-blocks of a Variational Auto Encoder

The conditional distribution

- Recall that a Bayesian network specification includes the conditional probability distribution $p(x_i \mid \mathrm{pa}(x_i))$ for each variable X_i.
- Typically the CPD is assumed to belong to some distributional family out of convenience — e.g., to obtain conjugacy.
- Deep Bayesian models allow the CPDs to be represented by DNNs.
- Since **inference is optimization**, we can adjust the parameters of the DNN and do inference in the model **interchangeably** while learning.

Building-blocks of a Variational Auto Encoder

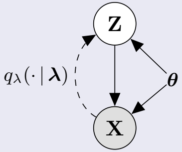
The conditional distribution

- Recall that a Bayesian network specification includes the conditional probability distribution $p(x_i \mid \mathrm{pa}(x_i))$ for each variable X_i.
- Typically the CPD is assumed to belong to some distributional family out of convenience — e.g., to obtain conjugacy.
- Deep Bayesian models allow the CPDs to be represented by DNNs.
- Since **inference is optimization**, we can adjust the parameters of the DNN and do inference in the model **interchangeably** while learning.

The model structure

- Bayesian models often leverage **latent variables**. These are variables $\mathbf{Z}$ that are unobserved, yet influence the observed variables $\mathbf{X}$.
- We therefore consider a model of two components:
  - $\mathbf{Z}$ follows some distribution $p_\theta(\mathbf{z} \mid \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$.
  - $\mathbf{X} \mid \mathbf{Z}$ follows some distribution $p_\theta(\mathbf{x} \mid g_\theta(\mathbf{z}))$ where $g_\theta(\mathbf{z})$ is a function represented by a deep neural network.
- In VAE lingo, $\mathbf{Z}$ in a **coded** version of $\mathbf{X}$. Therefore, $p_\theta(\mathbf{x} \mid g_\theta(\mathbf{z}))$ is the **decoder** model. Similarly, the process $\mathbf{X} \rightsquigarrow \mathbf{Z}$ is the **encoder**.

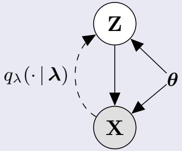
Model of interest



- We assume parametric distributions $p_\theta(\mathbf{z} \mid \boldsymbol{\theta})$ and $p_\theta\left(\mathbf{x} \mid \mathbf{z}, \boldsymbol{\theta}\right) = p_\theta\left(\mathbf{x} \mid g_{\boldsymbol{\theta}}(\mathbf{z})\right)$, where $g_{\boldsymbol{\theta}}(\cdot)$ for instance may be represented using a deep neural network.
- No further assumptions made about the generative model.
- We want to learn $\boldsymbol{\theta}$ to maximize the model's fit to the data-set $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.
- We cannot calculate $p(\mathbf{z} \mid \mathbf{x})$ analytically, so define the variational approximation $q_\lambda(\mathbf{z} \mid \mathbf{x}, \boldsymbol{\lambda})$. It will be represented by a DNN with parameters $\boldsymbol{\lambda}$.

The Variational Auto Encoder (VAE)

Model of interest



- We assume parametric distributions $p_\theta(\mathbf{z} \mid \boldsymbol{\theta})$ and $p_\theta\left(\mathbf{x} \mid \mathbf{z}, \boldsymbol{\theta}\right) = p_\theta\left(\mathbf{x} \mid g_{\boldsymbol{\theta}}(\mathbf{z})\right)$, where $g_{\boldsymbol{\theta}}(\cdot)$ for instance may be represented using a deep neural network.
- No further assumptions made about the generative model.
- We want to learn $\boldsymbol{\theta}$ to maximize the model's fit to the data-set $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.
- We cannot calculate $p(\mathbf{z} \mid \mathbf{x})$ analytically, so define the variational approximation $q_\lambda(\mathbf{z} \mid \mathbf{x}, \boldsymbol{\lambda})$. It will be represented by a DNN with parameters $\boldsymbol{\lambda}$.

Obvious strategy:

Optimize $\mathcal{L}(q)$ to choose $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$, where

$$\mathcal{L}(q) = -\mathbb{E}_{q_\lambda}\left[\log \frac{q_\lambda(\mathbf{z} \mid \mathbf{x}, \boldsymbol{\lambda})}{p_\theta(\mathbf{z}, \mathbf{x} \mid \boldsymbol{\theta})}\right]$$

Remember:

- We will parameterize $p_\theta(\mathbf{x} \mid \mathbf{z}, \boldsymbol{\theta})$ as a DNN with inputs $\mathbf{z}$ and weights defined by $\boldsymbol{\theta}$;
- ... and $q_\lambda(\mathbf{z} \mid \mathbf{x}, \boldsymbol{\lambda})$ as a DNN with inputs $\mathbf{x}$ and weights defined by $\boldsymbol{\lambda}$.

ELBO for VAEs

We will now look at ELBO **for a single observation** $\mathbf{x}_i$, and later maximize the sum of these contributions.

For a given $\mathbf{x}_i$ we get

$$\begin{aligned}
\mathcal{L}(\mathbf{x}_i) &= -\mathbb{E}_{q_\lambda}\left[\log \frac{q_\lambda(\mathbf{z} \,|\, \mathbf{x}_i, \boldsymbol{\lambda})}{p_\theta(\mathbf{z}, \mathbf{x}_i \,|\, \boldsymbol{\theta})}\right] \\
&= -\mathbb{E}_{q_\lambda}\left[\log q_\lambda(\mathbf{z} \,|\, \mathbf{x}_i, \boldsymbol{\lambda})\right] + \left\{\mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{z})\right] + \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}_i \,|\, \mathbf{z}, \boldsymbol{\theta})\right]\right\} \\
&= -\,\mathrm{KL}\left(q_\lambda(\mathbf{z} \,|\, \mathbf{x}_i, \boldsymbol{\lambda}) || p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}_i \,|\, \mathbf{z}, \boldsymbol{\theta})\right]
\end{aligned}$$

The two terms penalizes:

- ... a posterior over $\mathbf{z}$ far from the prior $p_\theta(\mathbf{z})$
- ...and poor reconstruction ability – averaged over $q_\lambda(\mathbf{z} \,|\, \mathbf{x}_i, \boldsymbol{\lambda})$

Calculating the ELBO terms

$$\mathcal{L}(\mathbf{x}_i) = -\mathrm{KL}\left(q_\lambda(\mathbf{z} \,|\, \mathbf{x}_i, \boldsymbol{\lambda}) || p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}_i \,|\, \mathbf{z}, \boldsymbol{\theta})\right]$$

- The KL-term is dependent on the distributional families of $p_\theta(\mathbf{z})$ and $q_\lambda(\mathbf{z} \,|\, \mathbf{x}_i, \boldsymbol{\lambda})$.
  - One can assume a simple shape, like:
    - $p_\theta(\mathbf{z})$ being Gaussian with zero mean and isotropic covariance;
    - $q_\lambda(z_\ell \,|\, \mathbf{x}_i, \boldsymbol{\lambda})$ is a Gaussian with mean and variance determined by a DNN.
  - Simplicity is **not required** as long as the KL can be calculated (numerically).

Calculating the ELBO terms

$$\mathcal{L}(\mathbf{x}_i) = -\operatorname{KL}\left(q_\lambda(\mathbf{z}\,|\,\mathbf{x}_i,\boldsymbol{\lambda})||p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}_i\,|\,\mathbf{z},\boldsymbol{\theta})\right]$$

- The KL-term is dependent on the distributional families of $p_\theta(\mathbf{z})$ and $q_\lambda(\mathbf{z}\,|\,\mathbf{x}_i,\boldsymbol{\lambda})$.
  - One can assume a simple shape, like:
    - $p_\theta(\mathbf{z})$ being Gaussian with zero mean and isotropic covariance;
    - $q_\lambda(z_\ell\,|\,\mathbf{x}_i,\boldsymbol{\lambda})$ is a Gaussian with mean and variance determined by a DNN.
  - Simplicity is **not required** as long as the KL can be calculated (numerically).
- The reconstruction term involves two separate operations:
  - For a given $\mathbf{z}$ evaluate the log-probability of the data-point $\mathbf{x}_i$, $\log p_\theta(\mathbf{x}_i\,|\,\mathbf{z},\boldsymbol{\theta})$. The distribution is parameterized by a DNN, getting its weights from $\boldsymbol{\theta}$.
  - The expectation $\mathbb{E}_{q_\lambda}[\cdot]$ is approximated by a random sample that we generate from $q_\lambda(\mathbf{z}\,|\,\mathbf{x}_i,\boldsymbol{\lambda})$:

$$\mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}_i\,|\,\mathbf{z},\boldsymbol{\theta})\right] \approx \frac{1}{M}\sum_{j=1}^{M}\log p_\theta\left(\mathbf{x}_i\,|\,\tilde{\mathbf{z}}_{i,j},\boldsymbol{\theta}\right),$$

    where $\tilde{\mathbf{z}}_{i,j}$ are samples from $q_\lambda(\cdot\,|\,\mathbf{x}_i,\boldsymbol{\lambda})$. Typically M is small (e.g., $M=1$).

Increased $\mathcal{L}(\mathbf{x}_i)$

Sample from $q_\lambda(\cdot \mid \mathbf{x}_i, \boldsymbol{\lambda})$

$\{\tilde{\mathbf{z}}_{i,\cdot}\}$

Approximate

$\mathbb{E}_{q_\lambda}\left[p_\theta(\mathbf{x}_i \mid \mathbf{z}, \boldsymbol{\theta})\right]$

Update $\boldsymbol{\theta}, \boldsymbol{\lambda}$ wrt. $\nabla_{\boldsymbol{\theta},\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}_i)$

Fun with MNIST – The model

- Each $\mathbf{x}_i$ is a binary vector of 784 values – **binarized** and **flattened** MNIST.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit ("0" – "9")

- Each $\mathbf{x}_i$ is a binary vector of 784 values – **binarized** and **flattened** MNIST.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit (“0” – “9”)

- Encoding is – for now – in **two** dimensions. A priori $\mathbf{Z}_i \sim p_\theta(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.
- The approximate expectation in the ELBO is calculated using $M = 1$ sample per data-point.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$ is a $256 + 64$ neural net with ReLU units.
  - The 64 outputs go through a linear layer to define $\boldsymbol{\mu}_\lambda(\mathbf{x}_i)$ and $\log \boldsymbol{\Sigma}_\lambda(\mathbf{x}_i)$.
  - Finally, $q_\lambda(\mathbf{z}_i \mid \mathbf{x}_i, \boldsymbol{\lambda}) = \mathcal{N}(\boldsymbol{\mu}_\lambda(\mathbf{x}_i), \boldsymbol{\Sigma}_\lambda(\mathbf{x}_i))$.

$\mathbf{x}_i : 784$ dim $\xrightarrow{\text{ReLU}}$ Hidden, 256-d $\xrightarrow{\text{ReLU}}$ Hidden, 64-d

Hidden, 64-d $\xrightarrow{\text{Linear}}$ $\log \boldsymbol{\Sigma}_\lambda(\mathbf{x}_i)$, 2-d $\rightarrow$ $q_\lambda(\mathbf{z}_i) = \mathcal{N}(\boldsymbol{\mu}_\lambda(\mathbf{x}_i), \boldsymbol{\Sigma}_\lambda(\mathbf{x}_i))$, 2-d

Hidden, 64-d $\xrightarrow{\text{Linear}}$ $\boldsymbol{\mu}_\lambda(\mathbf{x}_i)$, 2-d $\rightarrow$ $q_\lambda(\mathbf{z}_i) = \mathcal{N}(\boldsymbol{\mu}_\lambda(\mathbf{x}_i), \boldsymbol{\Sigma}_\lambda(\mathbf{x}_i))$, 2-d

- Each $\mathbf{x}_i$ is a binary vector of 784 values – **binarized** and **flattened** MNIST.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit ("0" – "9")

- Encoding is – for now – in **two** dimensions. A priori $\mathbf{Z}_i \sim p_\theta(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.
- The approximate expectation in the ELBO is calculated using $M = 1$ sample per data-point.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$ is a $256 + 64$ neural net with ReLU units.
  - The 64 outputs go through a linear layer to define $\boldsymbol{\mu}_\lambda(\mathbf{x}_i)$ and $\log \boldsymbol{\Sigma}_\lambda(\mathbf{x}_i)$.
  - Finally, $q_\lambda(\mathbf{z}_i \mid \mathbf{x}_i, \boldsymbol{\lambda}) = \mathcal{N}(\boldsymbol{\mu}_\lambda(\mathbf{x}_i), \boldsymbol{\Sigma}_\lambda(\mathbf{x}_i))$.
- The **decoder network** $\mathbf{Z} \rightsquigarrow \mathbf{X}$ is a $64 + 256$ neural net with ReLU units.
  - The 256 outputs go through a linear layer to define logit $(\mathbf{p}_\theta(\mathbf{z}_i))$.
  - Then $p_\theta(\mathbf{x}_i \mid \mathbf{z}_i, \boldsymbol{\theta})$ is Bernoulli with parameters $\mathbf{p}_\theta(\mathbf{z}_i)$.

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

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.

Training examples (after 500 epoch)

Examples from a separate test-set

Disentangled representations

What is a disentangled representation?

Representation learning:

- Representation learning is to find a mapping $r_{\boldsymbol{\theta}} : \mathcal{X} \mapsto \mathcal{R} \subseteq \mathbb{R}^d$ parameterized by $\boldsymbol{\theta}$, where $r_{\boldsymbol{\theta}}(\mathbf{x})$ is the **representation** of an observation $\mathbf{x}$.
- The underlying **manifold assumption** declares that while observations may be observed in an high-dimensional space $\mathcal{X}$, it (mostly) lives on a (smooth) low-dimensional manifold. The goal is to represent an image of this manifold on $\mathcal{R}$.

Supervised: The representation is an intermediate step towards, e.g., a classification – for instance an intermediate layer in a DNN.

Unsupervised: The representation is created without necessarily knowing its purpose later on. **This will be our focus.**

What is a disentangled representation?

Representation learning:

- Representation learning is to find a mapping $r_{\boldsymbol{\theta}} : \mathcal{X} \mapsto \mathcal{R} \subseteq \mathbb{R}^d$ parameterized by $\boldsymbol{\theta}$, where $r_{\boldsymbol{\theta}}(\mathbf{x})$ is the **representation** of an observation $\mathbf{x}$.
- The underlying **manifold assumption** declares that while observations may be observed in an high-dimensional space $\mathcal{X}$, it (mostly) lives on a (smooth) low-dimensional manifold. The goal is to represent an image of this manifold on $\mathcal{R}$.

  Supervised: The representation is an intermediate step towards, e.g., a classification – for instance an intermediate layer in a DNN.

  Unsupervised: The representation is created without necessarily knowing its purpose later on. **This will be our focus.**

Disentangled representations:

Assume that an object $\mathbf{x}$ is determined by **"data generative factors"**, e.g., what objects are in a picture, rotation, illumination, etc. Now, a disentangled representation should capture these factors.

Modularity: A single dim of $r_{\boldsymbol{\theta}}(\mathbf{x})$ encodes no more than one data generative factor.

Compactness: Each data generative factor is encoded by just one dim of $r_{\boldsymbol{\theta}}(\mathbf{x})$.

Explicitness: All data generative factors can be decoded from $r_{\boldsymbol{\theta}}(\mathbf{x})$ by a (linear) transformation.

Disentangled representations

Positives:

A disentangled representation $r_{\boldsymbol{\theta}}(\cdot)$ holds the promise to be ...

- interpretable
- robust towards noise
- useful for efficient learning of downstream tasks
- a representation for masking out “private” generating factors (gender, race, ...)

... and the idea has already been used for, e.g., fair machine learning, concept learning from video, domain adaption/transfer, ...

Disentangled representations

Positives:

A disentangled representation $r_{\boldsymbol{\theta}}(\cdot)$ holds the promise to be ...

- interpretable
- robust towards noise
- useful for efficient learning of downstream tasks
- a representation for masking out "private" generating factors (gender, race, ...)

... and the idea has already been used for, e.g., fair machine learning, concept learning from video, domain adaption/transfer, ...

Negative: Non-identifiability

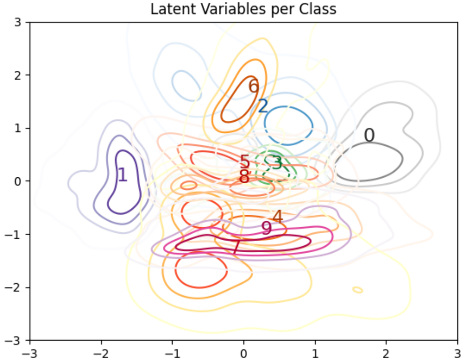
- Assume $\mathbf{z} = r_{\boldsymbol{\theta}}(\mathbf{x})$ is a disentangled representation according to the true generating factors of $p(\mathbf{x})$.
- We can create another representation $\mathbf{z}' = r_{\boldsymbol{\theta}'}(\mathbf{x})$ so that
  - $\mathbf{z}$ and $\mathbf{z}'$ are entangled
  - $\mathbf{z}$ and $\mathbf{z}'$ imply the same $p(\mathbf{x})$
- Observing only samples from $p(\mathbf{x})$, it is impossible to determine which of $r_{\boldsymbol{\theta}}(\cdot)$ and $r_{\boldsymbol{\theta}'}(\cdot)$ is the the better disentangled representation.

$\Rightarrow$ To be useful, $r_{\boldsymbol{\theta}}(\cdot)$ must be chosen **based on inductive bias**.

Using a VAE for representation learning

- The VAE is a **deep generative model**
- ...but can also be seen as a (probabilistic) **representation learning setup**:

$$r_{\boldsymbol{\lambda}}(\mathbf{x}) \sim q_{\lambda}(\cdot \,|\, \mathbf{x}, \boldsymbol{\lambda}).$$

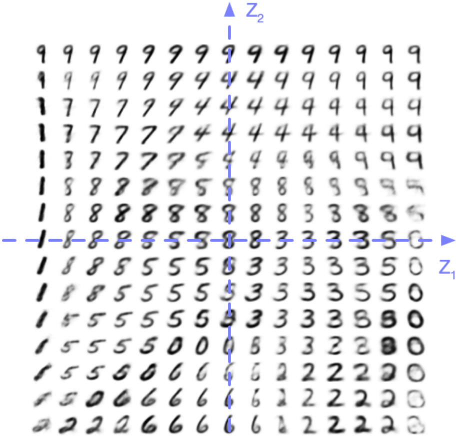


Checking the VAE: Interpretation of encoding space

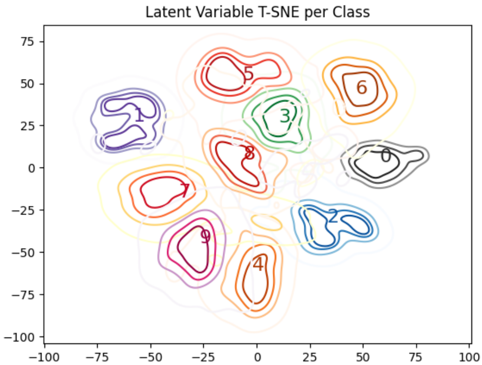
Investigations into the representation

Look for **modularity**, **compactness**, and **explicitness**:

- Imagine trips through $\mathbf{Z}$-space, and calculate $\mathbb{E}_{p_\theta}[\mathbf{X} \mid \mathbf{z}]$ for different values of $\mathbf{z}$: Does each dimension "make sense"? Can they be interpreted independently?
- Lots of **quantitative disentanglement metrics** exist as well.



Same results, but with high-dimensional encoding space

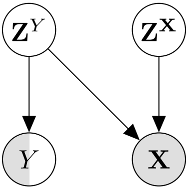


$\mathbb{E}_{p_\theta}[\mathbf{X} \mid \mathbf{z}]$-trajectories

Setup:

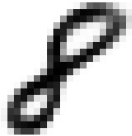
- Same VAE model, but now $\mathbf{Z}$ has 50 dims.
- Class-specific posterior $q_{\boldsymbol{\lambda}}(\mathbf{Z} = \mathbf{z} \mid \mathbf{X} = \mathbf{x})$ t-SNE'd down to 2 dims.
- **Animations:** $\mathbb{E}_{p_\theta}[\mathbf{X} \mid \mathbf{z}]$ varying a single latent dim (keeping the others at 0).
- Representations are interesting, but unclear if they are **disentangled**.

Extension: Semi-supervised learning

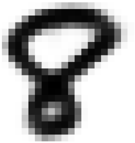


- The MNIST data consists of the **images** ($\mathbf{X}$) and their **classes** (which digit, Y).
  - We have so far not used the information in Y.
  - Now we will assume Y is at least sometimes observed.
- The code is extended to have two (a priori) **independent** parts: $\mathbf{Z}^{\mathbf{X}}$ and $\mathbf{Z}^Y$.
  - Both $\mathbf{Z}^{\mathbf{X}}$ and $\mathbf{Z}^Y$ contribute to define $\mathbf{X}$
  - Only $\mathbf{Z}^Y$ determines the class Y.
- The idea is that $\mathbf{Z}^{\mathbf{X}}$ is freed up to describe **class-independent features**.
  - We hope that $\mathbf{Z}^{\mathbf{X}}$ will capture globally valid and disentangled features describing something like “writing-style”.

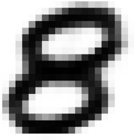
Conditional generation



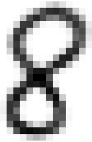
Z_0: "Slant"



Z_6: "Top heaviness"



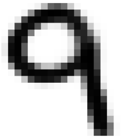
Z_{37}: "Width"



Z_{47}: "Pen thickness"

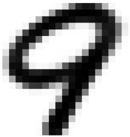
Process:

- Sample $\mathbf{z}_0^Y \sim p_\theta(\mathbf{z}^Y)$.
- Let $\mathbf{z}^{\mathbf{X}} = \mathbf{0}$ in all dims except j; vary $z_j^{\mathbf{X}}$. Calculate $\mathbb{E}_{p_\theta}\left[\mathbf{X} \,|\, \mathbf{z}^{\mathbf{X}}, \mathbf{z}_0^Y\right]$.

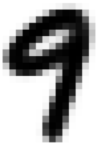


Z_0: “Slant”

Z_6: “Top heaviness”



Z_{37}: “Width”



Z_{47}: “Pen thickness”

Process:

- Sample $\mathbf{z}_0^Y \sim p_\theta(\mathbf{z}^Y)$.
- Let $\mathbf{z}^{\mathbf{X}} = \mathbf{0}$ in all dims except j; vary $z_j^{\mathbf{X}}$. Calculate $\mathbb{E}_{p_\theta}\left[\mathbf{X} \,|\, \mathbf{z}^{\mathbf{X}}, \mathbf{z}_0^Y\right]$.

A loose argument based on investigating the objective

The ELBO includes the penalty term $\mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}_i) || p(\mathbf{z})\right)$. If $q(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and k is the dimensionality of $\mathbf{z}$, then

$$\mathrm{KL}\left(q||p\right) = \frac{1}{2}\left[\boldsymbol{\mu}^{\mathsf{T}}\boldsymbol{\mu} + \mathrm{trace}(\boldsymbol{\Sigma}) - k \boxed{-\log|\boldsymbol{\Sigma}|}\right].$$

If $\boldsymbol{\Sigma}$'s diagonal is fixed, $\mathrm{KL}\left(q||p\right)$ is minimized for **independent** Z_j's.

β-VAE introduces a β to get a new loss (Std. VAE has $\beta = 1$):

$$\mathcal{L}(\mathbf{x}_i) = \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}_i \mid \boldsymbol{\theta})\right] - \boxed{\beta \cdot \mathrm{KL}\left(q_\lambda(\mathbf{z} \mid \mathbf{x}_i) || p_\theta(\mathbf{z})\right)}$$

The VAE's drivers for disentangled representations

A loose argument based on investigating the objective

The ELBO includes the penalty term $\mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}_i) || p(\mathbf{z})\right)$. If $q(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and k is the dimensionality of $\mathbf{z}$, then

$$\mathrm{KL}\left(q || p\right) = \frac{1}{2}\left[\boldsymbol{\mu}^{\mathsf{T}}\boldsymbol{\mu} + \text{trace}(\boldsymbol{\Sigma}) - k \; - \log|\boldsymbol{\Sigma}|\right].$$

If $\boldsymbol{\Sigma}$'s diagonal is fixed, $\mathrm{KL}\left(q || p\right)$ is minimized for **independent** Z_j's.

β-VAE introduces a β to get a new loss (Std. VAE has $\beta = 1$):

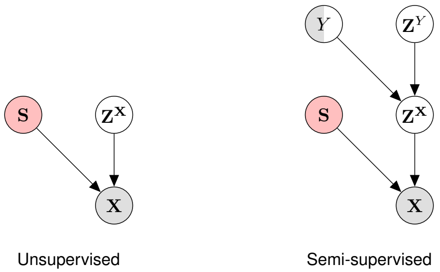
$$\mathcal{L}(\mathbf{x}_i) = \mathbb{E}_{q_\lambda}\left[\log p_\theta(\mathbf{x}_i \mid \boldsymbol{\theta})\right] - \; \beta \cdot \mathrm{KL}\left(q_\lambda(\mathbf{z} \mid \mathbf{x}_i) || p_\theta(\mathbf{z})\right)$$

There are **many** other loss-surgery approaches, too...

Dissecting the VAE objective reveals it includes the term

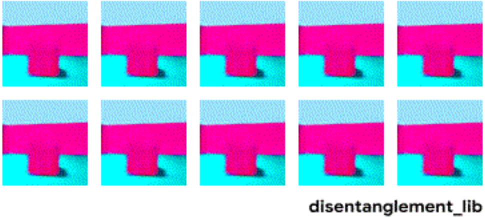
$$\mathrm{KL}\left(q(\mathbf{z} \mid \mathbf{x}_i) || \prod_{j=1}^{k} q_j(z_j \mid \mathbf{x}_i)\right),$$

where $q_j(z_j \mid \mathbf{x}_i)$ is the marginal variational distribution for Z_j. β-TCVAE multiplies that part of the loss with a $\beta \geq 1$.



- **Setup:** The data, $\mathbf{x}$, contains some **private ("secret") information** $\mathbf{s}$ (race, gender, political leaning, religion, ...)
- **Unsupervised (Left):** Find a representation $\mathbf{z}^{\mathbf{X}}$ that **cleans out** all traces of $\mathbf{s}$.
- **Semisupervised (Right):** Ensure that $\mathbf{z}^{\mathbf{X}}$ is **informative for the class** Y; supply $\mathbf{z}^{Y}$ for further **downstream processing**. Note that $\mathbf{z}^{Y}$ may now "loose" some information about y (as $\mathbf{z}^{\mathbf{X}}$ did about $\mathbf{s}$).
- **Learning objective:** Optimize ELBO, similarly as for VAE, but always conditioned on the private information. Add extra penalty if $\mathbf{S}$ is **predictable** from $\mathbf{z}^{\mathbf{X}}$.

Ready-made implementations and models



disentanglement_lib:

- **Open-source library** for learning disentangled representation by Google (`https://github.com/google-research/disentanglement_lib`)
- Implements a number of **benchmark models** (like β-VAE, β-TCVAE, ...), and relevant disentanglement metrics .
- Supplies standard **datasets**.
- Includes 10.800 **pre-learned models** ("*Reproducing these experiments requires approximately 2.52 GPU years*")

Probabilistic Programming Languages

Pyro and other PPLs

Pyro

Pyro (`https://pyro.ai`) is a Python library for probabilistic modeling and inference, integrated with Pytorch.

Modeling:
- Directed graphical models
- Neural networks (via `torch.nn`)
- ...

Inference:
- Variational inference
- MCMC – including Hamiltonian Monte Carlo, NUTS
- ...

... and there are also many other possibilities

- Tensorflow is integrating probabilistic thinking (`tensorflow_probability`)
- `pyMC3` is another Python-based alternative using Theano
- `turing.jl` is a new alternative for Julia
- ...

Example: Setup for VAE in Pyro

Setup:

- We define the generative model using a `model` (which is a stochastic function); use `obs=<data>` to condition on observations
- The `guide` defines how unobserved variables can be sampled (and thereby define our q-distribution)
- Learning optimizes parameterizations (typically using high-level abstractions like `pyro.infer.SVI` and `pyro.infer.TraceELBO`)
- Inference is done by gradient descent using an optimizer from Pytorch, e.g. `torch.optim.Adam`

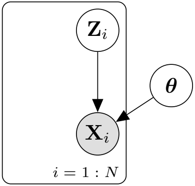
Code to define the optimization:

```
svi = SVI(model, guide, optimizer=Adam({lr: 1e-3}), loss=TraceELBO)
```

Code to do the actual training:

```
for xs in batches:
   losses.append(svi.step(xs))
```

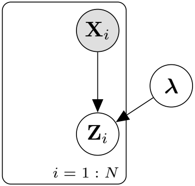
Generative model (`model`): $\mathbf{Z} \rightsquigarrow \mathbf{X}$




```python
# The `plate` defines a loop over the observations
with pyro.plate("data"):
    # Sample latents from the pre-defined prior distribution
    zs = pyro.sample("z",
        dist.Normal(
            torch.zeros(batch_size, self.z_dim),
            torch.ones(batch_size, self.z_dim)
        ).to_event(1))

    # Score the data (x) using the `handwriting style` (z),
    # where `decoder` is a neural network.
    # Note the conditioning using `obs=xs`
    probs = self.decoder.forward(zs)
    pyro.sample("x",
        dist.Bernoulli(probs).to_event(1), obs=xs)
```

Variational model (`guide`): $\mathbf{X} \rightsquigarrow \mathbf{Z}$




```python
# The `plate` defines a loop over the observations
with pyro.plate("data"):
    # Sample (and score) the latent `handwriting-style`
    # with the variational distribution
    # q(z|x) = Normal(loc(x),scale(x))
    loc, scale = self.encoder.forward(xs)
    pyro.sample("z", dist.Normal(loc, scale).to_event(1))
```

Conclusions

Summer school 2021

If you want to learn more about these things:

Nordic Probabilistic AI School,
June 14th – 18th, 2021
`https://probabilistic.ai`

Applications open soon!

Deep Learning + Probabilistic modelling = ♡: More robust AI models, resilience towards missing/adversarial examples, uncertainty awareness, ...

Variational Bayes: VB is a deterministic alternative to sampling for **approximate inference in Bayesian models**.

- VB seeks the model $q_{\lambda}(\mathbf{z} \mid \boldsymbol{\lambda}_{\mathbf{x}}) \in \mathcal{Q}$ closest to the (unattainable) posterior $p(\mathbf{z} \mid \mathbf{x})$ in terms of a **KL divergence**.
- BBVI performs **inference using gradient techniques**.

VAEs: A Variational Auto Encoders is an example of a probabilistic AI model.

- It is a deep **generative** model.
- Can be as a representation learner, as it generates "**encodings**" from examples.
- **Disentangled** representations are better for explainability, transparency, and other niceties.

Probabilistic Programming Languages: PPLs are programming languages to describe probabilistic models and perform inference in them.

- **Pyro** is a PPL built on top of Pytorch, and which supports several inference techniques, including BBVI, MCMC.
- Several **alternatives** exist as well.