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 $p(y_k \mid \text{pa}(y_k))$.

The full model is defined as $p(y) = p(y_1, \ldots, y_n) = \prod_{i=1}^{n} p(y_i \mid \text{pa}(y_i))$.

Markov properties $\Leftrightarrow$ Factorisation property.
Our focus yesterday was on **approximate Inference**: How to efficiently approximate \( p(z \mid x) \) by a simpler \( q(z \mid x) \).

- Looking for a “good” approximation means minimizing \( \text{KL} (q(z) \mid\mid p(z \mid x)) \).
  - 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 \( Q = \{ \text{All distributions that factorize} \} \).
  - Each local distribution \( q_i(z_i \mid \lambda_i) \) needs some pre-selected distributional family.
  - ... while we get to play with the parameters \( \lambda_i \).
  - 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

\[ Z \sim \mathcal{N}(0, I) \]

\[ X | z \sim \mathcal{N}(\mu + W^T z, \Sigma) \]

- The FA model posits that the data \( X \) can be generated from **independent factors** \( Z \) plus some sensor-noise: \( X | z = \mu + W^T z + \epsilon; \epsilon \sim \mathcal{N}(0, \Sigma) \).
- **Simple algorithms** to find estimators \( \hat{\mu}, \hat{W}, \) and \( \hat{\Sigma} \), and closed form expression for \( p(z | x) \) (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse \( W \) eases the interpretation.

**Variational Auto-Encoders**

- From Factor Analysis to Variational Auto-Encoders (VAEs) allow the distribution \( p(x | z) \) to be arbitrarily complex – represented by a DNN.
- We no longer have analytic estimators for model parameters, cannot easily calculate \( p(z | x) \), and it is therefore harder to interpret the factors \( Z \).
- **Why that name?** VAEs are called auto-encoders because we can train them by "re-creating" inputs via \( x \rightarrow p(z | x) \rightarrow z \rightarrow p(x | z) \rightarrow \hat{x} \) (and expect to see \( x \approx \hat{x} \)).
- It is a variational auto-encoder since we use the variational objective while learning.
The factor analysis model, and an extension

\[ Z \sim \mathcal{N}(0, I) \]

\[ X \mid z \sim \mathcal{N}(\mu + W^T z, \Sigma) \]

- The FA model posits that the data \( X \) can be generated from **independent factors** \( Z \) plus some sensor-noise: \( X \mid z = \mu + W^T z + \epsilon; \epsilon \sim \mathcal{N}(0, \Sigma) \).
- **Simple algorithms** to find estimators \( \hat{\mu}, \hat{W}, \) and \( \hat{\Sigma} \), and closed form expression for \( p(z \mid x) \) (which is still a Gaussian).
- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse \( W \) eases the interpretation.

**Example: Grades**

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

\[
\mathbb{E}[Z \mid x] = \begin{bmatrix}
0.25 & 0.25 & 0.25 & 0.25 \\
0.50 & 0  & 0.35 & 0.15 \\
\end{bmatrix}.
\]

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

Probabilistic AI – Lecture 2

Variational Auto-Encoders
The factor analysis model, and an extension

\[ Z \sim \mathcal{N}(0, I) \]

\[ X | z \sim \mathcal{N}(\mu + W^T z, \Sigma) \]

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

- **Simple algorithms** to find estimators \( \hat{\mu}, \hat{W}, \) and \( \hat{\Sigma} \), and closed form expression for \( p(z | x) \) (which is still a Gaussian).

- The idea is that the factors can be **interpreted** and used for **downstream tasks**. Typically a sparse \( W \) 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 \( Z \) to \( X \) has to be linear.
The factor analysis model, and an extension

\[ Z_1 \rightarrow X_1 \]
\[ Z_2 \rightarrow X_2 \]
\[ Z_1 \rightarrow X_3 \]
\[ Z_2 \rightarrow X_4 \]

VAE: \( Z \sim \) “Whatever”, typically still \( \mathcal{N}(0, I) \)

VAE: \( X \mid z \sim \) “Whatever”

- The FA model posits that the data \( X \) can be generated from independent factors \( Z \) plus some sensor-noise: \( X \mid z = \mu + W^\top z + \epsilon; \epsilon \sim \mathcal{N}(0, \Sigma) \).
- Simple algorithms to find estimators \( \hat{\mu}, \hat{W}, \) and \( \hat{\Sigma} \), and closed form expression for \( p(z \mid x) \) (which is still a Gaussian).
- The idea is that the factors can be interpreted and used for downstream tasks. Typically a sparse \( W \) eases the interpretation.

From Factor Analysis to Variational Auto-Encoders

VAEs allow the distribution \( p(x \mid z) \) to be arbitrarily complex – represented by a DNN. We no longer have analytic estimators for model parameters, cannot easily calculate \( p(z \mid x) \), and it is therefore harder to interpret the factors \( Z \).
The factor analysis model, and an extension

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

**Simple algorithms** to find estimators $\hat{\mu}$, $\hat{W}$, and $\hat{\Sigma}$, and closed form expression for $p(z \mid x)$ (which is still a Gaussian).

**The idea is that the factors can be interpreted and used for downstream tasks.** Typically a sparse $W$ eases the interpretation.

From Factor Analysis to Variational Auto Encoders

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

Why that name?

VAEs are called **auto-encoders** because we can train them by “re-creating” inputs via the process $x \xrightarrow{p(z \mid x)} z \xrightarrow{p(x \mid z)} \hat{x}$ (and expect to see $x \approx \hat{x}$).
The factor analysis model, and an extension

- The FA model posits that the data $X$ can be generated from independent factors $Z$ plus some sensor-noise: $X | z = \mu + W^T z + \epsilon; \epsilon \sim \mathcal{N}(0, \Sigma)$.
- Simple algorithms to find estimators $\hat{\mu}$, $\hat{W}$, and $\hat{\Sigma}$, and closed form expression for $p(z | x)$ (which is still a Gaussian).
- The idea is that the factors can be interpreted and used for downstream tasks. Typically a sparse $W$ eases the interpretation.

From Factor Analysis to Variational Auto Encoders

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

Why that name?

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

It is a variational auto-encoder since we use the variational objective while learning.
The conditional distribution

- Recall that a Bayesian network specification includes the conditional probability distribution $p(x_i \mid pa(x_i))$ 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 \( p(x_i \mid pa(x_i)) \) 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 \( Z \) that are unobserved, yet influence the observed variables \( X \).
- We therefore consider a model of two components:
  - \( Z \) follows some distribution \( p_{\theta}(z \mid \theta) \) parameterized by \( \theta \).
  - \( X \mid Z \) follows some distribution \( p_{\theta}(x \mid g_{\theta}(z)) \) where \( g_{\theta}(z) \) is a function represented by a deep neural network.
- In VAE lingo, \( Z \) in a coded version of \( X \). Therefore, \( p_{\theta}(x \mid g_{\theta}(z)) \) is the decoder model. Similarly, the process \( X \sim Z \) is the encoder.
The Variational Auto Encoder (VAE)

Model of interest

- We assume parametric distributions $p_\theta(z \mid \theta)$ and $p_\theta(x \mid z, \theta) = p_\theta(x \mid g_\theta(z))$, where $g_\theta(\cdot)$ for instance may be represented using a deep neural network.
- No further assumptions made about the generative model.
- We want to learn $\theta$ to maximize the model’s fit to the data-set $\mathcal{D} = \{x_1, \ldots, x_N\}$.
- We cannot calculate $p(z \mid x)$ analytically, so define the variational approximation $q_\lambda(z \mid x, \lambda)$. It will be represented by a DNN with parameters $\lambda$. 

Obvious strategy: Optimize $L(q)$ to choose $\lambda$ and $\theta$, where $L(q) = -E_{q_\lambda(z \mid x, \lambda)}[\log q_\lambda(z \mid x, \lambda)p_\theta(z, x \mid \theta)]$.
We assume parametric distributions $p_{\theta}(z | \theta)$ and $p_{\theta}(x | z, \theta) = p_{\theta}(x | g_{\theta}(z))$, where $g_{\theta}(\cdot)$ for instance may be represented using a deep neural network.

No further assumptions made about the generative model.

We want to learn $\theta$ to maximize the model’s fit to the data-set $D = \{x_1, \ldots, x_N\}$.

We cannot calculate $p(z | x)$ analytically, so define the variational approximation $q_{\lambda}(z | x, \lambda)$. It will be represented by a DNN with parameters $\lambda$.

**Obvious strategy:**
Optimize $L(q)$ to choose $\lambda$ and $\theta$, where

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

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

The Variational Auto Encoder (VAE)
We will now look at ELBO for a single observation $x_i$, and later maximize the sum of these contributions.

For a given $x_i$ we get

$$
\mathcal{L}(x_i) = -\mathbb{E}_{q_\lambda} \left[ \log \frac{q_\lambda(z | x_i, \lambda)}{p_\theta(z, x_i | \theta)} \right] \\
= -\mathbb{E}_{q_\lambda} [\log q_\lambda(z | x_i, \lambda)] + \left\{ \mathbb{E}_{q_\lambda} [\log p_\theta(z)] + \mathbb{E}_{q_\lambda} [\log p_\theta(x_i | z, \theta)] \right\} \\
= -KL(q_\lambda(z | x_i, \lambda) || p_\theta(z)) + \mathbb{E}_{q_\lambda} [\log p_\theta(x_i | z, \theta)]
$$

The two terms penalizes:

- ... a posterior over $z$ far from the prior $p_\theta(z)$
- ... and poor reconstruction ability – averaged over $q_\lambda(z | x_i, \lambda)$
Calculating the ELBO terms

\[ \mathcal{L}(x_i) = -\text{KL} (q_\lambda(z | x_i, \lambda) \| p_\theta(z)) + \mathbb{E}_{q_\lambda} [\log p_\theta(x_i | z, \theta)] \]

- The **KL-term** is dependent on the distributional families of \( p_\theta(z) \) and \( q_\lambda(z | x_i, \lambda) \).
  - One can assume a simple shape, like:
    - \( p_\theta(z) \) being Gaussian with zero mean and isotropic covariance;
    - \( q_\lambda(z | x_i, \lambda) \) 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

\[ \mathcal{L}(x_i) = -KL\left(q_\lambda(z \mid x_i, \lambda) \Vert p_\theta(z)\right) + \mathbb{E}_{q_\lambda}[\log p_\theta(x_i \mid z, \theta)] \]

- The **KL-term** is dependent on the distributional families of \( p_\theta(z) \) and \( q_\lambda(z \mid x_i, \lambda) \).
  - One can assume a simple shape, like:
    - \( p_\theta(z) \) being Gaussian with zero mean and isotropic covariance;
    - \( q_\lambda(z_\ell \mid x_i, \lambda) \) 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 \( z \) evaluate the log-probability of the data-point \( x_i \), \( \log p_\theta(x_i \mid z, \theta) \). The distribution is parameterized by a DNN, getting its weights from \( \theta \).
  - The expectation \( \mathbb{E}_{q_\lambda}[\cdot] \) is approximated by a random sample that we generate from \( q_\lambda(z \mid x_i, \lambda) \):
    \[
    \mathbb{E}_{q_\lambda}[\log p_\theta(x_i \mid z, \theta)] \approx \frac{1}{M} \sum_{j=1}^{M} \log p_\theta(x_i \mid \tilde{z}_{i,j}, \theta),
    \]
    where \( \tilde{z}_{i,j} \) are samples from \( q_\lambda(\cdot \mid x_i, \lambda) \). Typically \( M \) is small (e.g., \( M = 1 \)).
ELBO for VAEs

Sample from $q_{\lambda}(\cdot | x_i, \lambda)$

Increased $\mathcal{L}(x_i)$

Update $\theta, \lambda$ wrt. $\nabla_{\theta, \lambda} \mathcal{L}(x_i)$

$\mathbb{E}_{q_{\lambda}} [p_{\theta}(x_i | z, \theta)]$

Approximate

$\{\tilde{z}_i, \cdot \}$
Each $x_i$ is a binary vector of 784 values – **binarized** and **flattened** MNIST.

When seen as a $28 \times 28$ array, each $x_i$ is a picture of a handwritten digit (“0” – “9”).
Fun with MNIST – The model

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

![MNIST digits](image)

- Encoding is – for now – in **two** dimensions. A priori $\mathbf{z}_i \sim p_0(\mathbf{z}_i) = \mathcal{N}(0, \mathbf{I}_2)$.
- The approximate expectation in the ELBO is calculated using $M = 1$ sample per data-point.
- The **encoder network** $\mathbf{X} \xrightarrow{} \mathbf{Z}$ is a 256 + 64 neural net with ReLU units.
  - The 64 outputs go through a linear layer to define $\mu_\lambda(\mathbf{x}_i)$ and $\log \Sigma_\lambda(\mathbf{x}_i)$.
  - Finally, $q_\lambda (\mathbf{z}_i \mid \mathbf{x}_i, \lambda) = \mathcal{N}(\mu_\lambda(\mathbf{x}_i), \Sigma_\lambda(\mathbf{x}_i))$. 

![Encoder network diagram](image)
Fun with MNIST – The model

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

![Image with handwritten digits 5 7 4 4 2 6 3 5 9 2]

- Encoding is – for now – in **two** dimensions. A priori $Z_i \sim p_\theta(z_i) = \mathcal{N}(0_2, I_2)$.
- The approximate expectation in the ELBO is calculated using $M = 1$ sample per data-point.
- The **encoder network** $X \rightarrow Z$ is a $256 + 64$ neural net with ReLU units.
  - The 64 outputs go through a linear layer to define $\mu_\lambda(x_i)$ and $\log \Sigma_\lambda(x_i)$.
  - Finally, $q_\lambda(z_i \mid x_i, \lambda) = \mathcal{N}(\mu_\lambda(x_i), \Sigma_\lambda(x_i))$.
- The **decoder network** $Z \rightarrow X$ is a $64 + 256$ neural net with ReLU units.
  - The 256 outputs go through a linear layer to define logit ($p_\theta(z_i)$).
  - Then $p_\theta(x_i \mid z_i, \theta)$ is Bernoulli with parameters $p_\theta(z_i)$.

$$z_i : 2 \text{ dim} \xrightarrow{\text{ReLU}} \text{Hidden, 64-d} \xrightarrow{\text{ReLU}} \text{Hidden, 256-d} \xrightarrow{\text{Linear}} \text{logit}(p_i), 784-d \xrightarrow{} p_\theta(x_i \mid z_i) = \text{Bernoulli}(p_i), 784-d$$
Trying to reconstruct $\mathbf{x}$ by $E_{p\theta}[\mathbf{X} | \mathbf{Z} = E_{q\lambda}[\mathbf{Z} | \mathbf{x}_i]]$

An initial indication of performance:

1. For some $\mathbf{x}_0$, calculate $\mathbf{z}_0 \leftarrow E_{q\lambda}[\mathbf{Z} | \mathbf{X} = \mathbf{x}_0]$
2. ... and $\tilde{\mathbf{x}} \leftarrow E_{p\theta}[\mathbf{X} | \mathbf{Z} = \mathbf{z}_0]$.
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 $r_\theta : \mathcal{X} \mapsto \mathcal{R} \subseteq \mathbb{R}^d$ parameterized by $\theta$, where $r_\theta(x)$ is the representation of an observation $x$.

- 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 \( r_\theta : \mathcal{X} \mapsto \mathcal{R} \subseteq \mathbb{R}^d \) parameterized by \( \theta \), where \( r_\theta(x) \) is the representation of an observation \( x \).
- 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 \( x \) 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 \( r_\theta(x) \) encodes no more than one data generative factor.
  **Compactness:** Each data generative factor is encoded by just one dim of \( r_\theta(x) \).
  **Explicitness:** All data generative factors can be decoded from \( r_\theta(x) \) by a (linear) transformation.
Disentangled representations

Positives:

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

$\Rightarrow$ To be useful, $r_\theta(\cdot)$ must be chosen based on inductive bias.
Checking the VAE: Averaged distribution over $Z$ – per class

Using a VAE for representation learning

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

  $r_{\lambda}(x) \sim q_{\lambda}(\cdot \mid x, \lambda)$.
Investigations into the representation

Look for **modularity**, **compactness**, and **explicitness**:  
- Imagine trips through $Z$-space, and calculate $E_{p_{\theta}}[X | z]$ for different values of $z$: 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

**Setup:**

- Same VAE model, but now \( Z \) has 50 dims.
- Class-specific posterior \( q_{\lambda}(Z = z \mid X = x) \) t-SNE’d down to 2 dims.
- **Animations:** \( \mathbb{E}_{p_\theta}[X \mid z] \) varying a single latent dim (keeping the others at 0).
- Representations are interesting, but unclear if they are **disentangled**.
The MNIST data consists of the images $(X)$ 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: $Z^X$ and $Z^Y$.
- Both $Z^X$ and $Z^Y$ contribute to define $X$
- Only $Z^Y$ determines the class $Y$.

The idea is that $Z^X$ is freed up to describe class-independent features.
- We hope that $Z^X$ will capture globally valid and disentangled features describing something like “writing-style”.

Diagram:
- $Z^Y$ connected to $Y$
- $Z^X$ connected to $X$
- $Z^Y$ connected to $Z^X$
- $Z^X$ connected to $Z^Y$
Conditional generation

$Z_0$: “Slant”

$Z_6$: “Top heaviness”

$Z_{37}$: “Width”

$Z_{47}$: “Pen thickness”

**Process:**
- Sample $z^Y_0 \sim p_\theta(z^Y)$.
- Let $z^X = 0$ in all dims except $j$; vary $z^X_j$. Calculate $\mathbb{E}_{p_\theta} [X \mid z^X, z^Y_0]$. 
Conditional generation

$Z_0$: “Slant”  $Z_6$: “Top heaviness”

$Z_{37}$: “Width”  $Z_{47}$: “Pen thickness”

**Process:**
- Sample $z_0^Y \sim p_\theta(z^Y)$.
- Let $z^X = 0$ in all dims except $j$; vary $z_j^X$. Calculate $\mathbb{E}_{p_\theta} [X \mid z^X, z_0^Y]$.
A loose argument based on investigating the objective

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

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

If $\Sigma$’s diagonal is fixed, $\mathrm{KL} (q || p)$ is minimized for \textbf{independent} $Z_j$’s.

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

$$\mathcal{L}(x_i) = \mathbb{E}_{q\lambda} \left[ \log p_\theta(x_i | \theta) \right] - \beta \cdot \mathrm{KL} (q_\lambda(z | x_i) || p_\theta(z))$$
A loose argument based on investigating the objective

The ELBO includes the penalty term $\text{KL} (q(z \mid x_i) \| p(z))$. If $q(z) = \mathcal{N}(\mu, \Sigma)$, $p(z) = \mathcal{N}(0, I)$, and $k$ is the dimensionality of $z$, then

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

If $\Sigma$’s diagonal is fixed, $\text{KL} (q \| p)$ is minimized for independent $Z_j$’s.

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

$$
\mathcal{L}(x_i) = \mathbb{E}_{q_\lambda} [\log p_\theta(x_i \mid \theta)] - \beta \cdot \text{KL} (q_\lambda(z \mid x_i) \| p_\theta(z))
$$

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

Dissecting the VAE objective reveals it includes the term

$$
\text{KL} \left( q(z \mid x_i) \| \prod_{j=1}^k q_j(z_j \mid x_i) \right),
$$

where $q_j(z_j \mid x_i)$ is the marginal variational distribution for $Z_j$. $\beta$-TCVAE multiplies that part of the loss with a $\beta \geq 1$. 
Setup: The data, $x$, contains some private ("secret") information $s$ (race, gender, political leaning, religion, ...)

Unsupervised (Left): Find a representation $z^X$ that cleans out all traces of $s$.

Semisupervised (Right): Ensure that $z^X$ is informative for the class $Y$; supply $z^Y$ for further downstream processing. Note that $z^Y$ may now “loose” some information about $y$ (as $z^X$ did about $s$).

Learning objective: Optimize ELBO, similarly as for VAE, but always conditioned on the private information. Add extra penalty if $S$ is predictable from $z^X$. 
**disentanglement_lib:**

- **Open-source library** for learning disentangled representation by Google ([https://github.com/google-research/disentanglement_lib](https://github.com/google-research/disentanglement_lib))
- Implements a number of **benchmark models** (like $\beta$-VAE, $\beta$-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](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](https://www.tensorflow.org/probability))
- `pyMC3` is another Python-based alternative using Theano
- `turing.jl` is a new alternative for Julia
- …
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:**

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

**Code to do the actual training:**

```python
for xs in batches:
    losses.append(svi.step(xs))
```
**Generative model (model):** $Z \sim X$

```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=x`
    probs = self.decoder.forward(zs)
    pyro.sample("x",
        dist.Bernoulli(probs).to_event(1), obs=x)
```

**Variational model (guide):** $X \sim Z$

```python
# The `plate` defines a loop over the observations
with pyro.plate("data"):  
    # Sample (and score) the latent `handwriting-style`
    # with the variational distribution
    # $q(z|x) = \text{Normal}(\text{loc}(x), \text{scale}(x))$
    loc, scale = self.encoder.forward(xs)
    pyro.sample("z", dist.Normal(loc, scale).to_event(1))
```
Conclusions
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 $q_\lambda(z | \lambda, x) \in Q$ closest to the (unattainable) posterior $p(z | x)$ 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.