From shallow to deep latent variable models

Pierre-Alexandre Mattei

Inria, Université Côte d'Azur

pamattei.github.io
@pamattei
pierre-alexandre.mattei@inria.fr

Overview of todday’s lecture

Super short recap about last lecture

Latent variable models

Non-deep latent variable models

Approximate maximum likelihood for DLVMs

Deep learning as a building block in probabilistic models

Deep learning is a **general framework for function approximation.**

Deep learning as a building block in probabilistic models

Deep learning is a **general framework for function approximation.**

It can be used as a building block within probabilistic models by **using neural nets to model unknown functions.**

Deep learning as a building block in probabilistic models

Deep learning is a **general framework for function approximation.**

It can be used as a building block within probabilistic models by **using neural nets to model unknown functions.**

Last week's simple example was **binary classification:**

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \mathcal{B}(y|\pi_{\boldsymbol{\theta}}(\mathbf{x})) = \pi_{\boldsymbol{\theta}}(\mathbf{x})^y(1-\pi_{\boldsymbol{\theta}}(\mathbf{x}))^{1-y},$$

where **the function $\mathbf{x} \mapsto \pi_{\boldsymbol{\theta}}(\mathbf{x})$ is a neural net with parameters θ.** Why it it useful again for classification to use a neural net for $\pi_{\boldsymbol{\theta}}$?

Deep learning as a building block in probabilistic models

Deep learning is a **general framework for function approximation.**

It can be used as a building block within probabilistic models by **using neural nets to model unknown functions.**

Last week's simple example was **binary classification:**

$$p_{\boldsymbol{\theta}}(y|\mathbf{x}) = \mathcal{B}(y|\pi_{\boldsymbol{\theta}}(\mathbf{x})) = \pi_{\boldsymbol{\theta}}(\mathbf{x})^y(1-\pi_{\boldsymbol{\theta}}(\mathbf{x}))^{1-y},$$

where **the function $\mathbf{x} \mapsto \pi_{\boldsymbol{\theta}}(\mathbf{x})$ is a neural net with parameters $\boldsymbol{\theta}$.** Why it it useful again for classification to use a neural net for $\pi_{\boldsymbol{\theta}}$?

For many reasons! It can model arbitrarily complex decision boundaries, we can use prior information in the architecure (e.g. for image classification), we can do large-scale maximum likelihood training using stochastic gradient descent, leverage GPUs...

Overview of todday’s lecture

Super short recap about last lecture

Latent variable models

Non-deep latent variable models

Approximate maximum likelihood for DLVMs

Latent variable models

Let's assume that we have i.i.d. data $\mathbf{x}_1, ..., \mathbf{x}_n$ that live in a **high-dimensional space $\mathcal{X}$** (for example images of newspaper articles).

Latent variable models

Let's assume that we have i.i.d. data $\mathbf{x}_1, ..., \mathbf{x}_n$ that live in a **high-dimensional space** $\mathcal{X}$ (for example images of newspaper articles).

Often, it is reasonable to assume that **these data essentially depend on a few important factors of variation:**

- if we have images of faces: size of nose, color of hair, glasses or not...
- if we have newspaper articles: topics of the paper, style...
- if we have molecules: physical properties, geometrical shape...

Latent variable models

Let's assume that we have i.i.d. data $\mathbf{x}_1, ..., \mathbf{x}_n$ that live in a **high-dimensional space** $\mathcal{X}$ (for example images of newspaper articles).

Often, it is reasonable to assume that **these data essentially depend on a few important factors of variation:**

- if we have images of faces: size of nose, color of hair, glasses or not...
- if we have newspaper articles: topics of the paper, style...
- if we have molecules: physical properties, geometrical shape...

This assumption is the cornerstone of **latent variable models**, that assume that the data are governed by **unobserved random variables** $\mathbf{z}_1, ..., \mathbf{z}_n$ **that live in a low dimensional space** $\mathcal{Z}$ (for example $\mathbf{R}^2$). We can think of $\mathbf{z}_i$ as a **code** summarizing the essential factors of the data point $\mathbf{x}_i$.

Overview of todday’s lecture

Super short recap about last lecture

Latent variable models

Non-deep latent variable models

Approximate maximum likelihood for DLVMs

Factor analysis and probabilistic PCA

In this slide, we assume that the data are continuous (i.e. $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^p$).

Factor analysis and probabilistic PCA

In this slide, we assume that the data are continuous (i.e. $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^p$).

Factor analysis is probably one of the oldest latent variable models (studied since at least the 1940s). The generative process is:

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})$.

Factor analysis and probabilistic PCA

In this slide, we assume that the data are continuous (i.e. $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^p$).

Factor analysis is probably one of the oldest latent variable models (studied since at least the 1940s). The generative process is:

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})$.

Probabilistic PCA (PPCA, Tipping and Bishop, JRSSB, 1999) is a a slightly less general model that you already saw in the HD stats course:

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I}_p)$.

Factor analysis and probabilistic PCA

In this slide, we assume that the data are continuous (i.e. $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^p$).

Factor analysis is probably one of the oldest latent variable models (studied since at least the 1940s). The generative process is:

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})$.

Probabilistic PCA (PPCA, Tipping and Bishop, JRSSB, 1999) is a a slightly less general model that you already saw in the HD stats course:

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I}_p)$.

Related models for non-continuous data: topic models (text), Poisson PPCA (count data).
What is the geometric intuition?

From shallow to deep latent variable models

How do we transform this kind of model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I}_p)$.

...into "something deep"?

From shallow to deep latent variable models

How do we transform this kind of model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I}_p)$.

...into "something deep"?

We can replace the affine function $\mathbf{z} \mapsto \mathbf{W}\mathbf{z} + \mu$ by a neural net!

From shallow to deep latent variable models

How do we transform this kind of model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I}_p)$.

...into "something deep"?

We can replace the affine function $\mathbf{z} \mapsto \mathbf{W}\mathbf{z} + \mu$ by a neural net!

That's the key idea of **deep latent variable models (DLVMs)**, present in particular in both **variational autoencoders (VAEs)**, invented independently by Kingma and Welling (ICLR 2014) and Rezende, Mohamed & Wierstra (ICML 2014), and **generative adversarial networks (GANs)** (Goodfellow et al., NeurIPS 2014).

From shallow to deep latent variable models

How do we transform this kind of model

- $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_d)$,
- $\mathbf{x}_i | \mathbf{z}_i \sim \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I}_p)$.

...into "something deep"?

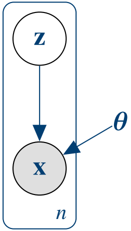
We can replace the affine function $\mathbf{z} \mapsto \mathbf{W}\mathbf{z} + \boldsymbol{\mu}$ by a neural net!

That's the key idea of **deep latent variable models (DLVMs)**, present in particular in both **variational autoencoders (VAEs)**, invented independently by Kingma and Welling (ICLR 2014) and Rezende, Mohamed & Wierstra (ICML 2014), and **generative adversarial networks (GANs)** (Goodfellow et al., NeurIPS 2014).

In this lecture, we'll focus on VAEs, but GANs are really quite similar.

Deep latent variable models (DLVMs)

Kingma and Welling (ICLR 2014), Rezende, Mohamed & Wierstra (ICML 2014), Goodfellow et al. (NeurIPS 2014)



Assume that $(\mathbf{x}_i, \mathbf{z}_i)_{i \leq n}$ are i.i.d. random variables driven by the model:

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) & \text{(observation model)} \end{cases}$$

where

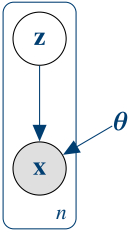
- $\mathbf{z} \in \mathbb{R}^d$ is the **latent** variable,
- $\mathbf{x} \in \mathcal{X}$ is the **observed** variable.

Deep latent variable models (DLVMs)

Kingma and Welling (ICLR 2014), Rezende, Mohamed & Wierstra (ICML 2014), Goodfellow et al. (NeurIPS 2014)

Assume that $(\mathbf{x}_i, \mathbf{z}_i)_{i \leq n}$ are i.i.d. random variables driven by the model:

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \Phi(\mathbf{x} \mid f_{\boldsymbol{\theta}}(\mathbf{z})) & \text{(observation model)} \end{cases}$$



where

- $\mathbf{z} \in \mathbb{R}^d$ is the **latent** variable,
- $\mathbf{x} \in \mathcal{X}$ is the **observed** variable,
- the function $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to H$ is a **(deep) neural network** called the **decoder**
- $(\Phi(\cdot \mid \boldsymbol{\eta}))_{\boldsymbol{\eta} \in H}$ is a parametric family called the **observation model**, usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

The role of the prior

As in regular factor analysis, the prior distribution of the latent variable is often an **isotropic Gaussian** $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}_d, \mathbf{I}_d)$.

The role of the prior

As in regular factor analysis, the prior distribution of the latent variable is often an **isotropic Gaussian** $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}_d, \mathbf{I}_d)$.

More complex, **learnable priors** have also been considered. For example, mixtures of K Gaussians:

$$p(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

where the parameters $\pi_1, ..., \pi_K, \boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, ...\boldsymbol{\Sigma}_K$ are learned.

The role of the prior

As in regular factor analysis, the prior distribution of the latent variable is often an **isotropic Gaussian** $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}_d, \mathbf{I}_d)$.

More complex, **learnable priors** have also been considered. For example, mixtures of K Gaussians:

$$p(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

where the parameters $\pi_1, ..., \pi_K, \boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, ...\boldsymbol{\Sigma}_K$ are learned.

Note that **this prior is not a prior in the Bayesian sense** (i.e., about parameter uncertainty).

The role of the observation model

The observation model $(\Phi(\cdot \mid \boldsymbol{\eta}))_{\boldsymbol{\eta} \in H}$ usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

Its parameters are the output of the decoder.

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{z})) & \text{(Gaussian observation model)} \end{cases}$$

The role of the observation model

The observation model $(\Phi(\cdot \mid \boldsymbol{\eta}))_{\boldsymbol{\eta} \in H}$ usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

Its parameters are the output of the decoder.

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{z})) & \text{(Gaussian observation model)} \end{cases}$$

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \mathcal{B}(\mathbf{x} \mid \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{z})) & \text{(Bernoulli observation model)} \end{cases}$$

The role of the observation model

The observation model $(\Phi(\cdot \mid \boldsymbol{\eta}))_{\boldsymbol{\eta}\in H}$ usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

Its parameters are the output of the decoder.

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{z})) & \text{(Gaussian observation model)} \end{cases}$$

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \mathcal{B}(\mathbf{x} \mid \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{z})) & \text{(Bernoulli observation model)} \end{cases}$$

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) = \mathrm{St}(\mathbf{x} \mid \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\nu}_{\boldsymbol{\theta}}(\mathbf{z})) & \text{(Student's t observation model)} \end{cases}$$

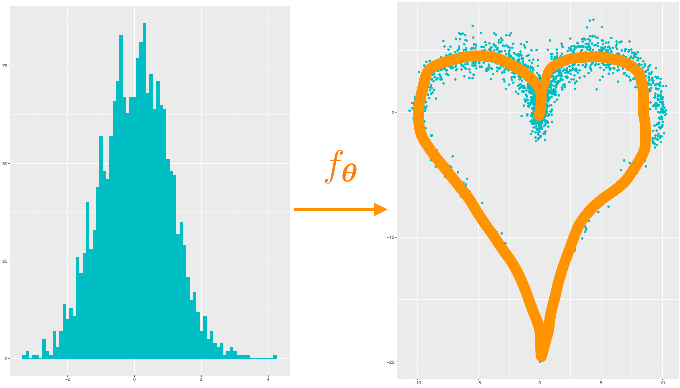
The role of the decoder

The role of the **decoder** $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to H$ is:

- to transform $\mathbf{z}$ (**the code**) into parameters $\eta = f_{\boldsymbol{\theta}}(\mathbf{z})$ of the observation model $\Phi(\cdot \mid \eta)$.
- The weights $\boldsymbol{\theta}$ of the **decoder** are learned.

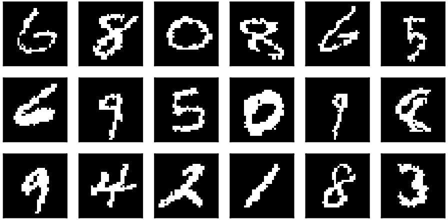
Simple non-linear decoder **($d = 1$, $p = 2$)**: $f_{\boldsymbol{\theta}}(z) = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{z})$ with, for all $z \in \mathbb{R}$,

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(z) = \left(10\sin(z)^3, 10\cos(z) - 10\cos(z)^4\right), \; \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\mathbf{z}) = \text{Diag}\left(\left(\frac{\sin(z)}{3z}\right)^2, \left(\frac{\sin(z)}{z}\right)^2\right).$$



Illustrative example of a DLVM

Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ binary MNIST



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0,1\}^{28\times 28}$

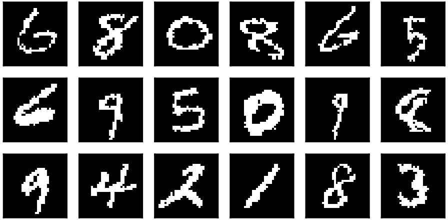
$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ x^{j,k} \sim \mathrm{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

Decoder network

$$f(\mathbf{z}) = \mathrm{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

Illustrative example of a DLVM

Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ binary MNIST



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0,1\}^{28\times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

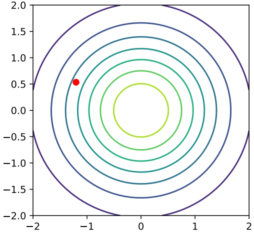
Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V}\tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

Generation

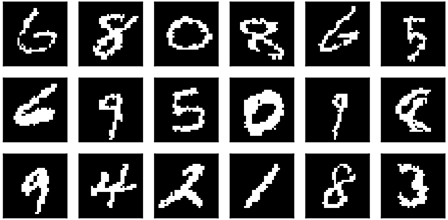
$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

$$\mathbf{z} = (-1.2033, 0.5340)$$



Illustrative example of a DLVM

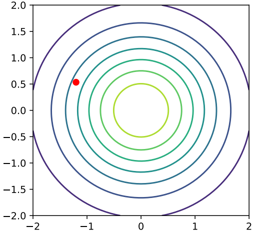
Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ binary MNIST



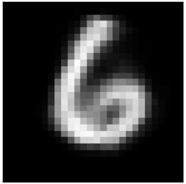
Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

$$\mathbf{z} = (-1.2033, 0.5340)$$



$$f(\mathbf{z})$$



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0,1\}^{28\times 28}$

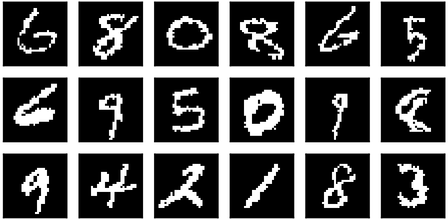
$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \beta)$$

Illustrative example of a DLVM

Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ binary MNIST



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0,1\}^{28\times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

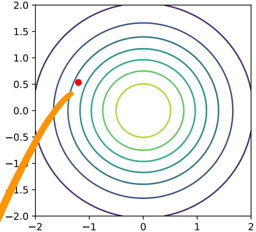
Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

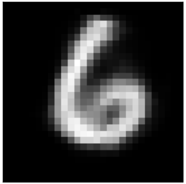
Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

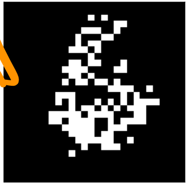
$$\mathbf{z} = (-1.2033, 0.5340)$$



$f(\mathbf{z})$

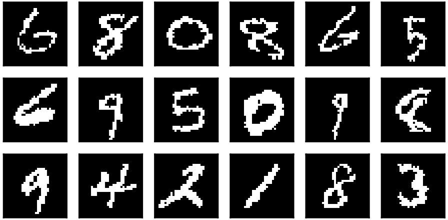


$x^{j,k} \sim \text{Bern}(f^{j,k}(\mathbf{z}))$



Illustrative example of a DLVM

Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ binary MNIST



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0,1\}^{28\times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

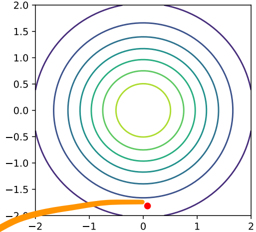
Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

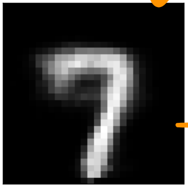
Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

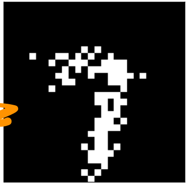
$$\mathbf{z} = (0.0791, -1.8165)$$



$f(z)$

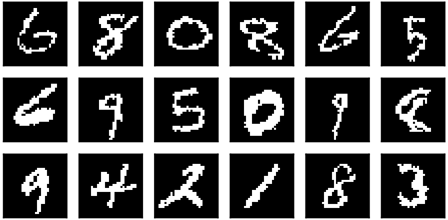


$x^{j,k} \sim \text{Bern}(f^{j,k}(z))$



Illustrative example of a DLVM

Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ binary MNIST



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0,1\}^{28\times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

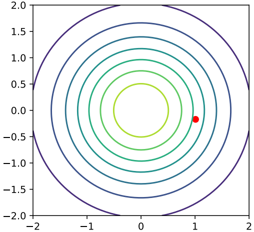
Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

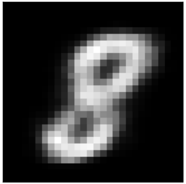
Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

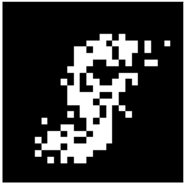
$$\mathbf{z} = (1.0097, -0.1711)$$



$f(z)$



$x^{j,k} \sim \text{Bern}(f^{j,k}(z))$



DLVMs applications: density estimation on MNIST

Rezende, Mohamed & Wierstra (ICML 2014)

Training data

Model samples

DLVMs applications: clustering on MNIST

Harchaoui, Mattei, Bouveyron & Almansa (2018)

$\mathbf{m}_k$

$\mathbf{m}_k + 0.5\mathbf{S}_k\mathbf{u}$

$\mathbf{m}_k + \mathbf{S}_k\mathbf{u}$

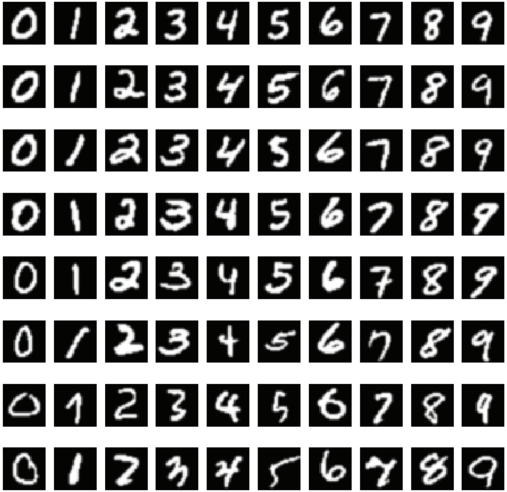
$\mathbf{m}_k + 1.5\mathbf{S}_k\mathbf{u}$

$\mathbf{m}_k + 2\mathbf{S}_k\mathbf{u}$

$\mathbf{m}_k + 2.5\mathbf{S}_k\mathbf{u}$

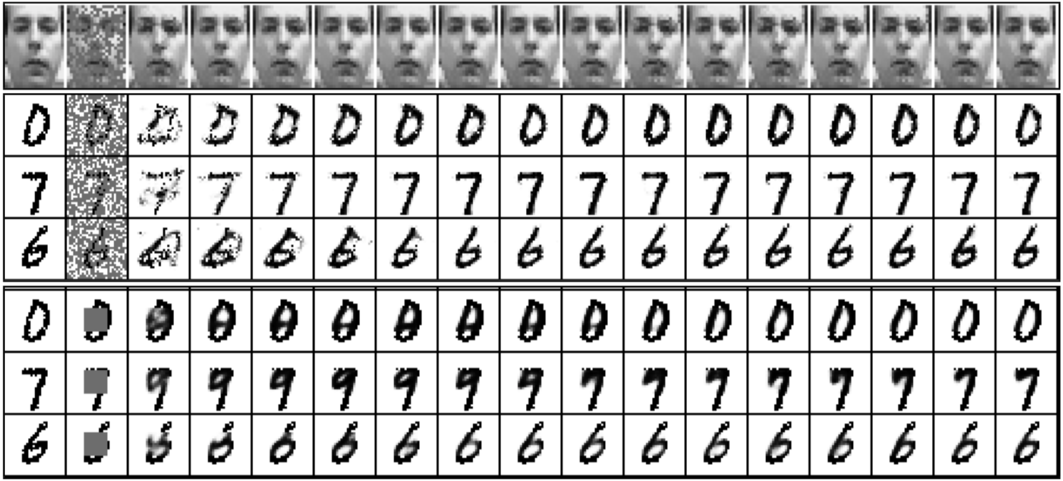
$\mathbf{m}_k + 3\mathbf{S}_k\mathbf{u}$

$\mathbf{m}_k + 3.5\mathbf{S}_k\mathbf{u}$



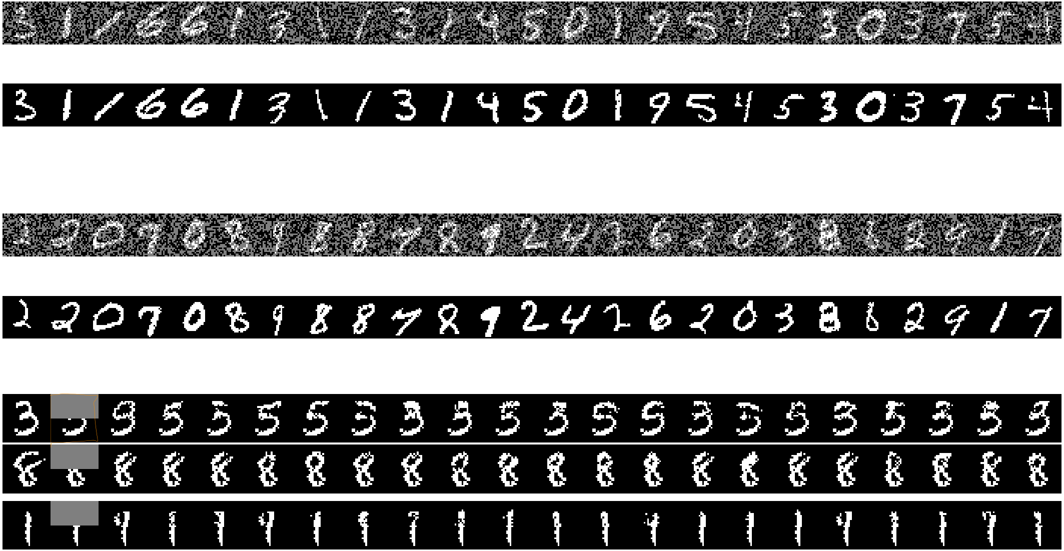
DLVMs applications: Data imputation

Rezende, Mohamed & Wierstra (ICML 2014)



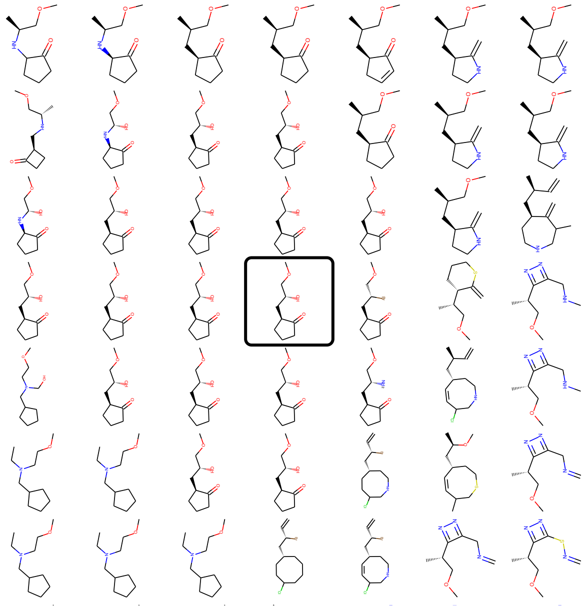
DLVMs applications: Data imputation

Mattei & Frellsen (ICML 2019)



DLVMs applications: Molecular design

Kusner, Paige, and Hernàndez-Lobato (2017)



DLVMs applications: Reinforcement learning

Ha & Schmidhuber (2018)

DLVMs applications: Reinforcement learning

Ha & Schmidhuber (2018)

Overview of todday’s lecture

Super short recap about last lecture

Latent variable models

Non-deep latent variable models

Approximate maximum likelihood for DLVMs

Learning DLVMs

Kingma & Welling (2014), Rezende et al. (2014)

Given a data matrix $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\intercal \in \mathcal{X}^n$, the **log-likelihood function** for a DLVM is

$$\ell(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i),$$

where

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

We would like to find a **MLE** $\hat{\theta} \in \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$.

:

Learning DLVMs

Kingma & Welling (2014), Rezende et al. (2014)

Given a data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\intercal \in \mathcal{X}^n$, the **log-likelihood function** for a DLVM is

$$\ell(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i),$$

where

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

We would like to find a **MLE** $\hat{\theta} \in \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$.

However, even with a simple output density $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$:

- $p_{\boldsymbol{\theta}}(\mathbf{x})$ is **intractable** rendering **MLE intractable**

Learning DLVMs

Kingma & Welling (2014), Rezende et al. (2014)

Given a data matrix $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^\intercal \in \mathcal{X}^n$, the **log-likelihood function** for a DLVM is

$$\ell(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i),$$

where

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

We would like to find a **MLE** $\hat{\theta} \in \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$.

However, even with a simple output density $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$:

- $p_{\boldsymbol{\theta}}(\mathbf{x})$ is **intractable** rendering **MLE intractable**
- $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$ is **intractable** rendering **EM intractable**

Learning DLVMs

Kingma & Welling (2014), Rezende et al. (2014)

Given a data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\intercal \in \mathcal{X}^n$, the **log-likelihood function** for a DLVM is

$$\ell(\boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\mathbf{X}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\mathbf{x}_i),$$

where

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

We would like to find a **MLE** $\hat{\theta} \in \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$.

However, even with a simple output density $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$:

- $p_{\boldsymbol{\theta}}(\mathbf{x})$ is **intractable** rendering **MLE intractable**
- $p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$ is **intractable** rendering **EM intractable**
- **stochastic EM is not scalable** to large n and moderate d.

The solution: amortised variational inference

A general and scalable framework to tackle these issues was proposed by Kingma & Welling (2014), Rezende et al. (2014), leading to the **variational autoencoder (VAE).**

The solution: amortised variational inference

A general and scalable framework to tackle these issues was proposed by Kingma & Welling (2014), Rezende et al. (2014), leading to the **variational autoencoder (VAE).**

Here, **I am going to derive this approach in a slightly different manner**, largely inspired by the following paper:

Burda, Grosse & Salakhutdinov (2016), *Importance weighted autoencoders*, ICLR 2016

The solution: amortised variational inference

A general and scalable framework to tackle these issues was proposed by Kingma & Welling (2014), Rezende et al. (2014), leading to the **variational autoencoder (VAE).**

Here, **I am going to derive this approach in a slightly different manner**, largely inspired by the following paper:

Burda, Grosse & Salakhutdinov (2016), *Importance weighted autoencoders*, ICLR 2016

The main idea is to use **Monte Carlo techniques** to approximate the intractable integrals

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}.$$

Aparté: Monte Carlo and importance sampling

Let's say we want to estimate an integral of the form

$$I = \int_\Omega f(x)p(x)dx,$$

where $f \geq 0$ and p is a density over a space Ω.

Aparté: Monte Carlo and importance sampling

Let's say we want to estimate an integral of the form

$$I = \int_{\Omega} f(x)p(x)dx,$$

where $f \geq 0$ and p is a density over a space Ω.

Simple Monte Carlo estimate: We sample $x_1, ..., .x_K \sim p$ and approximate

$$I \approx \frac{1}{K} \sum_{k=1}^{K} f(x_k) = \hat{I}_K.$$

A few properties:

$$\hat{I}_K \overset{a.s.}{\to} I, \;\; \mathbb{E}[\hat{I}_K] = I, \;\; \mathbb{V}[\hat{I}_K] = \frac{1}{K}\mathbb{V}[f(x_1)],$$

which sounds nice, but **the variance may be very large.**

Aparté: Monte Carlo and importance sampling

Importance sampling tries to improve this estimate by **sampling $x_1, ..., .x_K$ from another density q rather than p.** The trick is the following:

$$
\begin{aligned}
I &= \int_\Omega f(x)p(x)dx \\
&= \int_\Omega \frac{f(x)p(x)}{q(x)} q(x)dx \approx \frac{1}{K}\sum_{k=1}^{K} \frac{f(x_k)p(x_k)}{q(x_k)} = \hat{I}_K^q.
\end{aligned}
$$

Aparté: Monte Carlo and importance sampling

Importance sampling tries to improve this estimate by **sampling $x_1, \ldots, .x_K$ from another density q rather than p.** The trick is the following:

$$
\begin{aligned}
I &= \int_\Omega f(x)p(x)dx \\
&= \int_\Omega \frac{f(x)p(x)}{q(x)} q(x)dx \approx \frac{1}{K}\sum_{k=1}^{K} \frac{f(x_k)p(x_k)}{q(x_k)} = \hat{I}_K^q.
\end{aligned}
$$

As before, the estimate is **consistent and unbiased** (under the condition that q has heavier tails than p). **What about the variance?**

Aparté: Monte Carlo and importance sampling

Importance sampling tries to improve this estimate by **sampling $x_1, ..., .x_K$ from another density q rather than p.** The trick is the following:

$$
\begin{aligned}
I &= \int_\Omega f(x)p(x)dx \\
&= \int_\Omega \frac{f(x)p(x)}{q(x)} q(x)dx \approx \frac{1}{K}\sum_{k=1}^{K} \frac{f(x_k)p(x_k)}{q(x_k)} = \hat{I}_K^q.
\end{aligned}
$$

As before, the estimate is **consistent and unbiased** (under the condition that q has heavier tails than p). **What about the variance?**

If we choose $q^*(x) \propto f(x)p(x)$, which means $q^*(x) = f(x)p(x)/I$, then $\hat{I}_K^{q^*}$ **has zero variance!** But we can't do that because we don't know I...

Aparté: Monte Carlo and importance sampling

Importance sampling tries to improve this estimate by **sampling $x_1, \ldots, .x_K$ from another density q rather than p.** The trick is the following:

$$
\begin{aligned}
I &= \int_\Omega f(x)p(x)dx \\
&= \int_\Omega \frac{f(x)p(x)}{q(x)} q(x)dx \approx \frac{1}{K}\sum_{k=1}^{K} \frac{f(x_k)p(x_k)}{q(x_k)} = \hat{I}_K^q.
\end{aligned}
$$

As before, the estimate is **consistent and unbiased** (under the condition that q has heavier tails than p). **What about the variance?**

If we choose $q^*(x) \propto f(x)p(x)$, which means $q^*(x) = f(x)p(x)/I$, then $\hat{I}_K^{q^*}$ **has zero variance!** But we can't do that because we don't know I...

However, this still means that **importance sampling with a good q will work much better than simple MC.**

Back to DLVMs and their likelihood

We want to approximate

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

Back to DLVMs and their likelihood

We want to approximate

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

Idea: use importance sampling! Let $\mathbf{z}_{i1}, ..., \mathbf{z}_{iK}$ follow some proposal q_i:

$$\int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z})}{q_i(\mathbf{z}_{ik})}$$

Back to DLVMs and their likelihood

We want to approximate

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

Idea: use importance sampling! Let $\mathbf{z}_{i1}, ..., \mathbf{z}_{iK}$ follow some proposal q_i:

$$\int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z})}{q_i(\mathbf{z}_{ik})}$$

Let's say that we want to choose our **proposal in a parametric family** $(\Psi(\cdot|\kappa))_{\kappa \in \mathcal{K}}$ over $\mathbb{R}^d$ (e.g. Gaussians).

Back to DLVMs and their likelihood

We want to approximate

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

Idea: use importance sampling! Let $\mathbf{z}_{i1}, ..., \mathbf{z}_{iK}$ follow some proposal q_i:

$$\int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z})}{q_i(\mathbf{z}_{ik})}$$

Let's say that we want to choose our **proposal in a parametric family** $(\Psi(\cdot|\kappa))_{\kappa \in \mathcal{K}}$ over $\mathbb{R}^d$ (e.g. Gaussians).

Problem: we need to choose n **proposals** $q_1, ..., q_n$ (and n is usually large in deep learning...).

Back to DLVMs and their likelihood

We want to approximate

$$p_{\boldsymbol{\theta}}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) \mathbf{z}.$$

Idea: use importance sampling! Let $\mathbf{z}_{i1}, ..., \mathbf{z}_{iK}$ follow some proposal q_i:

$$\int_{\mathbb{R}^d} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}_{ik}) d\mathbf{z} \approx \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{z}_{ik}) p(\mathbf{z})}{q_i(\mathbf{z}_{ik})}$$

Let's say that we want to choose our **proposal in a parametric family** $(\Psi(\cdot|\kappa))_{\kappa \in \mathcal{K}}$ over $\mathbb{R}^d$ (e.g. Gaussians).

Problem: we need to choose n **proposals** $q_1, ..., q_n$ (and n is usually large in deep learning...).

Exercise: What would be the optimal, zero-variance choices for $q_1, ..., q_n$?

q*_i (z) = p(z I x_i)

Amortised variational inference

A solution: Amortised variational inference, **all the q_i will be defined together via a neural net!**

Amortised variational inference

A solution: Amortised variational inference, **all the q_i will be defined together via a neural net!**

Rationale: q_i needs to depends on $\mathbf{x}_i$, so we'll define it as a **conditional distribution parametrised by γ:**

$$q_i(\mathbf{z}) = q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i).$$

Amortised variational inference

A solution: Amortised variational inference, **all the q_i will be defined together via a neural net!**

Rationale: q_i needs to depends on $\mathbf{x}_i$, so we'll define it as a **conditional distribution parametrised by γ:**

$$q_i(\mathbf{z}) = q_\gamma(\mathbf{z}|\mathbf{x}_i).$$

How to parametrise this conditional distribution? The key idea is that **its parameters are the output of a neural net g_γ:**

$$q_\gamma(\mathbf{z}|\mathbf{x}_i) = \Psi(\mathbf{z}|g_\gamma(\mathbf{x}_i)).$$

Amortised variational inference

A solution: Amortised variational inference, **all the q_i will be defined together via a neural net!**

Rationale: q_i needs to depends on $\mathbf{x}_i$, so we'll define it as a **conditional distribution parametrised by γ:**

$$q_i(\mathbf{z}) = q_\gamma(\mathbf{z}|\mathbf{x}_i).$$

How to parametrise this conditional distribution? The key idea is that **its parameters are the output of a neural net g_γ:**

$$q_\gamma(\mathbf{z}|\mathbf{x}_i) = \Psi(\mathbf{z}|g_\gamma(\mathbf{x}_i)).$$

This neural net is called the **inference network** or **encoder**.

Amortised variational inference

All of this leads to the following approximation of the likelihood

$$\ell(\boldsymbol{\theta}) \approx \sum_{i=1}^{n} \mathbb{E}_{\mathbf{z}_{i1},\ldots,\mathbf{z}_{iK} \sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_{ik})p(\mathbf{z}_{ik})}{q_{\boldsymbol{\gamma}}(\mathbf{z}_{ik}|\mathbf{x}_i)} \right] = \mathcal{L}_K(\boldsymbol{\theta}, \gamma).$$

Amortised variational inference

All of this leads to the following approximation of the likelihood

$$\ell(\boldsymbol{\theta}) \approx \sum_{i=1}^{n} \mathbb{E}_{\mathbf{z}_{i1},\ldots,\mathbf{z}_{iK} \sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_{\boldsymbol{\gamma}}(\mathbf{z}_{ik}|\mathbf{x}_i)} \right] = \mathcal{L}_K(\boldsymbol{\theta}, \gamma).$$

Rather than maximising $\ell(\boldsymbol{\theta})$, **we'll maximise $\mathcal{L}_K(\theta, \gamma)$ using SGD** and the reparametrisation trick. But does it make sense to do that?

Amortised variational inference

All of this leads to the following approximation of the likelihood

$$\ell(\boldsymbol{\theta}) \approx \sum_{i=1}^{n} \mathbb{E}_{\mathbf{z}_{i1},\ldots,\mathbf{z}_{iK}\sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_{ik})p(\mathbf{z}_{ik})}{q_{\boldsymbol{\gamma}}(\mathbf{z}_{ik}|\mathbf{x}_i)} \right] = \mathcal{L}_K(\boldsymbol{\theta},\boldsymbol{\gamma}).$$

Rather than maximising $\ell(\boldsymbol{\theta})$, **we'll maximise $\mathcal{L}_K(\boldsymbol{\theta},\boldsymbol{\gamma})$ using SGD** and the reparametrisation trick. But does it make sense to do that?

It does make sense! For several reasons:

- $\mathcal{L}_K(\boldsymbol{\theta},\boldsymbol{\gamma})$ is a **lower bound of $\ell(\boldsymbol{\theta})$** (exercise !)
- The bounds get **tighter and tighter!**

$$\mathcal{L}_1(\boldsymbol{\theta},\boldsymbol{\gamma}) \leq \mathcal{L}_2(\boldsymbol{\theta},\boldsymbol{\gamma}) \leq \ldots \leq \mathcal{L}_K(\boldsymbol{\theta},\boldsymbol{\gamma}) \xrightarrow[K\to\infty]{} \ell(\boldsymbol{\theta}).$$

$\mathcal{L}_K(\boldsymbol{\theta},\boldsymbol{\gamma})$ is called the **importance weighted autoencoder (IWAE)** bound, and was introduced by Burda et al. (2016).