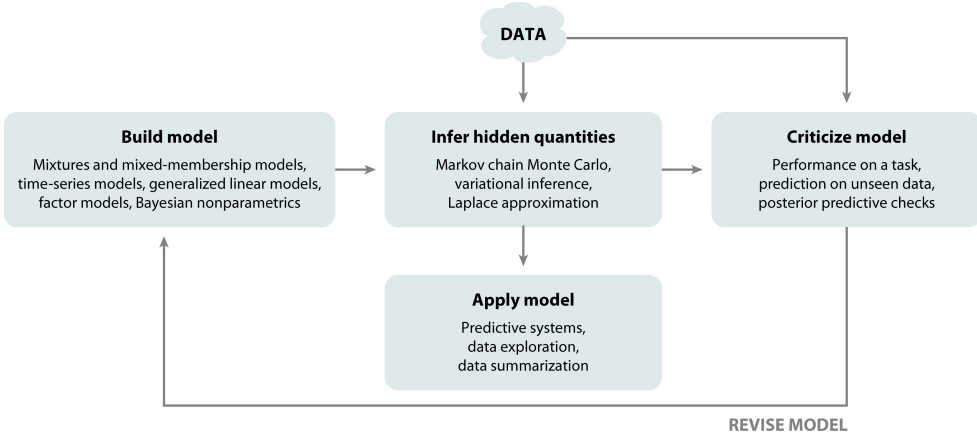
STATS271/371: Applied Bayesian Statistics

Factor Analysis, Variational Autoencoders, and Variational EM

Scott Linderman

May 12, 2021

Box's Loop



Blei DM. 2014.
Annu. Rev. Stat. Appl. 1:203–32

Blei, *Ann. Rev. Stat. App.* 2014.

Lap 6: Factor Analysis, Variational Autoencoders, and Variational EM

Let's walk before we run...

- **Model:** Factor Analysis
- **Algorithm:** Expectation Maximization

Next time

- **Model:** Nonlinear factor analysis
- **Algorithm:** Variational EM
- **Model+Algorithm:** Variational Autoencoders

The road map

Factor Analysis

Notation

Let,

- $\boldsymbol{x}_n \in \mathbb{R}^D$ denote the n-th observation,
- $\boldsymbol{z}_n \in \mathbb{R}^P$ denote the corresponding latent variables,
- $\boldsymbol{w}_d \in \mathbb{R}^P$ for $d = 1, \ldots, D$ denote the weights each dimension,
- $\boldsymbol{W} \in \mathbb{R}^{D \times P}$ denote the matrix with rows $\boldsymbol{w}_d$.
- $\sigma_d^2 \in \mathbb{R}_+$ for $d = 1, \ldots, D$ denote the variance for each dimension,
- $\boldsymbol{\Sigma} = \text{diag}([\sigma_1^2, \ldots, \sigma_D^2])$ denote the full covariance matrix, amd
- $\boldsymbol{\Theta} = (\boldsymbol{W}, \boldsymbol{\Sigma}) = \{\boldsymbol{w}_d, \sigma_d^2\}_{d=1}^D$ denote the set of all parameters.

The generative process

Factor analysis is a "linear Gaussian" latent variable model with the following generative process,

To generate N observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, each in $\mathbb{R}^D$,

1. Sample model parameters $\{\boldsymbol{w}_d, \sigma_d^2\}_{d=1}^D$ from their prior distribution.
2. For each observation $n = 1, \ldots, N$,
    a. Sample a **latent variable** from a multivariate Gaussian prior, $\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
    b. For each dimension $d = 1, \ldots, D$
        i. Apply a **linear transformation** to get the mean $\boldsymbol{w}_d^\top \boldsymbol{z}_n$
        ii. Sample the d-th coordinate of the **observation** $x_{n,d} \sim \mathcal{N}(\boldsymbol{w}_d^\top \boldsymbol{z}_n, \sigma_d^2)$.

Note: The last two steps are equivalent to sampling $\boldsymbol{x}_n \sim \mathcal{N}(\boldsymbol{W}\boldsymbol{z}_n, \boldsymbol{\Sigma})$ since $\boldsymbol{\Sigma}$ is diagonal.

The joint distribution

The joint probability is,

$$p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N, \boldsymbol{\Theta}) = p(\boldsymbol{\Theta}) \prod_{n=1}^{N} p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta})\, p(\boldsymbol{z}_n \mid \boldsymbol{\Theta}) \tag{1}$$

$$= p(\boldsymbol{\Theta}) \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{W}\boldsymbol{z}_n, \Sigma)\, \mathcal{N}(\boldsymbol{z}_n \mid \boldsymbol{0}, \boldsymbol{I}) \tag{2}$$

$$= p(\boldsymbol{\Theta}) \prod_{n=1}^{N} \prod_{d=1}^{D} \Big[\mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{w}_d^\top \boldsymbol{z}_n, \sigma_d^2) \Big] \mathcal{N}(\boldsymbol{z}_n \mid \boldsymbol{0}, \boldsymbol{I}) \tag{3}$$

We'll place an uninformative prior over the parameters,

$$p(\boldsymbol{\Theta}) = p(\{\boldsymbol{w}_d, \sigma_d^2\}_{d=1}^D) \propto \prod_{d=1}^{D} \sigma_d^{-2} \tag{4}$$

(Recall from Lap 1 that this is the uninformative limit of a normal inverse chi-squared distribution, $\text{Inv}{-}\chi^2(\sigma_d^2 \mid \nu, \tau^2)\, \mathcal{N}(\boldsymbol{w}_d \mid \boldsymbol{\mu}, \sigma_d^2 \boldsymbol{\Lambda}^{-1})$, with $\nu \to 0$ and $\boldsymbol{\Lambda} \to 0$.)

Warm up: Derive the Gibbs updates

Suppose we want to sample the posterior distribution, $p(\{\boldsymbol{w}_d, \sigma_d^2\}_{d=1}^D, \{\boldsymbol{z}_n\}_{n=1}^N \mid \{\boldsymbol{x}_n\}_{n=1}^N)$.

The posterior does not have a simple closed form, but as we learned, we can use MCMC algorithms to produce such samples.

The Gibbs sampler is one type of MCMC algorithm. It iteratively samples one (or more) variables from their conditional distribution, holding the rest fixed.

For the factor analysis model, this amounts to repeating the following steps *ad infinitum*:

1. for $n = 1, \ldots, N$, sample $\boldsymbol{z}_n \sim p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \{\boldsymbol{w}_d, \sigma_d^2\}_{d=1}^D)$
2. for $d = 1, \ldots, D$, sample $\boldsymbol{w}_d, \sigma_d^2 \sim p(\boldsymbol{w}_d, \sigma_d^2 \mid \{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N)$

(Note: we used the fact that $\boldsymbol{z}_n$ is conditionally indepedent of $\{\boldsymbol{x}_{n'}, \boldsymbol{z}_{n'}\}_{n' \neq n}$ given the parameters and $\boldsymbol{w}_d, \sigma_d^2$ are conditionally independent of $\{\boldsymbol{w}_{d'}, \sigma_{d'}^2\}_{d'=1}^D$ given the latent variables.)

Warm up: Derive the Gibbs updates II

Exercise: Derive the conditional distribution $\boldsymbol{z}_n \sim p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \{\boldsymbol{w}_d, \sigma_d^2\}_{d=1}^D)$.
Hint: it's easier to start with Eq. 2 and work with the matrices $\boldsymbol{W}$ and $\boldsymbol{\Sigma}$.

Warm up: Derive the Gibbs updates III

Exercise: Derive the conditional distribution $p(\boldsymbol{w}_d, \sigma_d^2 \mid \{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N)$.

Hint: Recall the posterior distribution for Bayesian linear regression model from Lap 1.

Should we parameters and latent variables the same?

Gibbs samplings aims for a full posterior distribution over both latent variables and parameters. This is great but perhaps overkill...

In probabilistic machine learning, we often talk about *learning* the parameters and *inferring* the parameters.

The distinction makes sense when there is more uncertainty about the latent variables $\boldsymbol{z}_n$, which are informed by a single data point, than about the parameters $\boldsymbol{\Theta}$, which depend on all the data.

In that case, we might only need a *point estimate* of the parameters and a *posterior* of the latent variables given the data and parameters.

Two goals

The **learning goal** is to find the parameters that **maximize the marginal probability of the data**,

$$\Theta^\star = \arg\max_{\Theta} p(\{\boldsymbol{x}_n\}_{n=1}^N, \Theta) \tag{5}$$

$$= \arg\max_{\Theta} p(\Theta) \prod_{n=1}^{N} \int p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \Theta)\, p(\boldsymbol{z}_n \mid \Theta)\, \mathrm{d}\boldsymbol{z}_n \tag{6}$$

The **inference goal** is to find the **posterior distribution of latent variables**,

$$p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \Theta) = \frac{p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \Theta)\, p(\boldsymbol{z}_n \mid \Theta)\, p(\Theta)}{p(\boldsymbol{x}_n, \Theta)} \tag{7}$$

$$= \frac{p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \Theta)\, p(\boldsymbol{z}_n \mid \Theta)}{\int p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \Theta)\, p(\boldsymbol{z}_n \mid \Theta)\, \mathrm{d}\boldsymbol{z}_n} \tag{8}$$

Both goals require an integral over $\boldsymbol{z}_n$, and this could potentially be hard!

Learning via gradient ascent

Let's start with the learning goal. We could try gradient ascent...

$$\nabla \log p(\{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta}) = \nabla \log p(\boldsymbol{\Theta}) + \sum_{n=1}^N \nabla \log \int p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta})\, p(\boldsymbol{z}_n \mid \boldsymbol{\Theta})\, \mathrm{d}\boldsymbol{z}_n \tag{9}$$

$$= \nabla \log p(\boldsymbol{\Theta}) + \sum_{n=1}^N \frac{\int \nabla p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta})\, p(\boldsymbol{z}_n \mid \boldsymbol{\Theta})\, \mathrm{d}\boldsymbol{z}_n}{\int p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta})\, p(\boldsymbol{z}_n \mid \boldsymbol{\Theta})\, \mathrm{d}\boldsymbol{z}_n} \tag{10}$$

For conjugate exponential family models, we can get a nice closed form gradient expression, but in general this is tricky.

Moreover, gradient ascent requires choices of a step size, and while that's not so difficult, it is a consideration.

Return of the ELBO

Idea: Use the ELBO to get a bound on the marginal probability and maximize that instead.

$$\log p(\{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta}) = \log p(\{\boldsymbol{x}_n\}_{n=1}^N \mid \boldsymbol{\Theta}) + \log p(\boldsymbol{\Theta}) \tag{11}$$

$$\geq \mathbb{E}_q\left[\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N \mid \boldsymbol{\Theta}) - \log q(\{\boldsymbol{z}_n\}_{n=1}^N)\right] + \log p(\boldsymbol{\Theta}) \tag{12}$$

$$\triangleq \mathscr{L}(q, \boldsymbol{\Theta}) \tag{13}$$

Here, I've explicitly written the ELBO as a function of both the distribution $q(\{\boldsymbol{z}_n\}_{n=1}^N)$ and the parameters $\boldsymbol{\Theta}$.

When is the ELBO tight?

Show that the ELBO is maximized when $q(\{\boldsymbol{z}_n\}_{n=1}^N) = p(\{\boldsymbol{z}_n\}_{n=1}^N \mid \{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta})$.

The Expectation-Maximization (EM) algorithm

The **expectation-maximization (EM) algorithm** maximizes this lower bound by iteratively optimizing wrt q and $\boldsymbol{\Theta}$.

1. Initialize parameters $\boldsymbol{\Theta}$
2. Repeat until $\boldsymbol{\Theta}$ (or the ELBO) converges:
   a. **E step:** Set

$$q(\{\boldsymbol{z}_n\}_{n=1}^N) = \arg\max_q \mathscr{L}(q, \boldsymbol{\Theta}) = p(\{\boldsymbol{z}_n\}_{n=1}^N \mid \{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta}). \tag{14}$$

   At this point, the ELBO is tight and $\mathscr{L}(q, \boldsymbol{\Theta}) = p(\{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta})$. Track this quantity!

   b. **M step:** Set

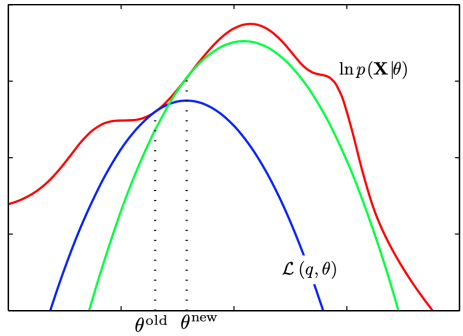
$$\boldsymbol{\Theta} = \arg\max_{\boldsymbol{\Theta}} \mathscr{L}(q, \boldsymbol{\Theta}) \tag{15}$$

$$= \arg\max_{\boldsymbol{\Theta}} \mathbb{E}_q[\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N \mid \boldsymbol{\Theta})] + \log p(\boldsymbol{\Theta}). \tag{16}$$

   The optimization in the second step can often be done in closed form.

The EM algorithm in pictures

Figure 9.14 The EM algorithm involves alternately computing a lower bound on the log likelihood for the current parameter values and then maximizing this bound to obtain the new parameter values. See the text for a full discussion.



From Bishop [2006].

E step for factor analysis

We've already derived the posterior,

$$p(\{\boldsymbol{z}_n\}_{n=1}^N \mid \{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta}) = \prod_{n=1}^N p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{\Theta}) \tag{17}$$

$$= \prod_{n=1}^N \mathcal{N}(\boldsymbol{z}_n \mid \boldsymbol{m}_n', \boldsymbol{C}_n') \tag{18}$$

where

$$\boldsymbol{J}_n' = \boldsymbol{I} + \boldsymbol{W}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{W} \qquad \boldsymbol{h}_n' = \boldsymbol{W}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{x}_n \tag{19}$$

$$\boldsymbol{C}_n' = \boldsymbol{J}_n'^{-1} \qquad \boldsymbol{m}_n' = \boldsymbol{J}_n'^{-1} \boldsymbol{h}_n'. \tag{20}$$

M step for factor analysis

Intuition: Given $\boldsymbol{z}$, this is basically a (Bayesian) linear regression problem.

As a function of $\boldsymbol{\Theta} = \{\boldsymbol{w}_d, \sigma_d^2\}_{d=1}^D$,

$$\mathscr{L}(q, \boldsymbol{\Theta}) = \mathbb{E}_q[\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N, \boldsymbol{\Theta})] \tag{21}$$

$$= \mathbb{E}_q\left[\sum_{n=1}^N \log p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta})\right] + \log p(\boldsymbol{\Theta}) + c \tag{22}$$

$$= -\frac{1}{2}\sum_{n=1}^N\sum_{d=1}^D\left(\log \sigma_d^2 + \mathbb{E}_{q(\boldsymbol{z}_n)}\left[\frac{1}{\sigma_d^2}(x_{n,d} - \boldsymbol{w}_d^\top \boldsymbol{z}_n)^2\right]\right) - \sum_{d=1}^D \log \sigma_d^2 \tag{23}$$

$$= \sum_{d=1}^D\left[-\frac{N+2}{2}\log \sigma_d^2 - \left\langle \frac{1}{2\sigma_d^2}, \sum_{n=1}^N x_{n,d}^2\right\rangle + \left\langle \frac{\boldsymbol{w}_d}{\sigma_d^2}, \sum_{n=1}^N \mathbb{E}_q[\boldsymbol{z}_n] x_{n,d}\right\rangle - \left\langle \frac{\boldsymbol{w}_d \boldsymbol{w}_d^\top}{2\sigma_d^2}, \sum_{n=1}^N \mathbb{E}_q[\boldsymbol{z}_n \boldsymbol{z}_n^\top]\right\rangle\right] \tag{24}$$

We recognize the term in the square brackets as the log of a normal inverse chi-squared distribution, the posterior in a Bayesian linear regression!

M step for factor analysis II

Recalling Lap 1, we have

$$\mathscr{L}(q, \Theta) = \mathbb{E}_q[\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N, \Theta)] \sum_{d=1}^{D} \left[\log \text{Inv}{-}\chi^2(\sigma_d^2 \mid \nu'_d, \tau'^2_d) + \log \mathcal{N}(\boldsymbol{w}_d \mid \boldsymbol{\mu}'_d, \sigma_d^2 \boldsymbol{\Lambda}'_d) \right] \quad (25)$$

where

$$\nu'_d = N \qquad \boldsymbol{\Lambda}'_d = \sum_{n=1}^{N} \mathbb{E}_q[\boldsymbol{z}_n \boldsymbol{z}_n^\top] \quad (26)$$

$$\boldsymbol{\mu}'_d = \boldsymbol{\Lambda}'^{-1}_d \sum_{n=1}^{N} \mathbb{E}_q[\boldsymbol{z}_n] x_{n,d} \qquad \tau'^2_d = \frac{1}{\nu'_d} \left(\sum_{n=1}^{N} x_{n,d}^2 - \boldsymbol{\mu}'^\top_d \boldsymbol{\Lambda}'_d \boldsymbol{\mu}'_d \right) \quad (27)$$

M step for factor analysis III

The maximum is achieved at the mode of this distribution,

$$\boldsymbol{w}_d \leftarrow \boldsymbol{\mu}'_d = \left(\sum_{n=1}^{N} \mathbb{E}_q[\boldsymbol{z}_n \boldsymbol{z}_n^\top]\right)^{-1} \left(\sum_{n=1}^{N} \mathbb{E}_q[\boldsymbol{z}_n] x_{n,d}\right) \tag{28}$$

$$\sigma_d^2 \leftarrow \frac{\nu'_d \tau'^2_d}{\nu'_d + 2} = \frac{1}{N+2}\left(\sum_{n=1}^{N} x_{n,d}^2 - \left(\sum_{n=1}^{N} \mathbb{E}_q[\boldsymbol{z}_n] x_{n,d}\right)^\top \left(\sum_{n=1}^{N} \mathbb{E}_q[\boldsymbol{z}_n \boldsymbol{z}_n^\top]\right)^{-1} \left(\sum_{n=1}^{N} \mathbb{E}_q[\boldsymbol{z}_n] x_{n,d}\right)\right) \tag{29}$$

It's exactly like in Lap 1, except here we have **expected sufficient statistics** of the latent variables instead of observed covariates.

The big picture

EM for latent variable models like factor analysis basically iterates between two steps:

1. Compute the posterior of the latent variables given the current parameters,
2. Find the parameters that maximize the expected log joint probability.

For many models (factor analysis, mixture models, etc.), these two steps can be done exactly.

In practice, EM often converges very quickly, but the objective is non-convex so you might need to try many random initializations!

Questions

- What if we had $\boldsymbol{x}_n \sim \mathcal{N}(f(\boldsymbol{z}_n, \Theta), \Sigma)$ for some nonlinear function f? Where did we use the linearity assumption?
- What can we do if the posterior needed for the E step is intractable?
- What if the M step doesn't have a closed form solution?

As we will see, variational autoencoders (VAEs) offer one set of answers to these questions.

Lap 6: Factor Analysis, Variational Autoencoders, and Variational EM

- **Model:** Factor Analysis
- **Algorithm:** Expectation Maximization
- **Model: Nonlinear factor analysis**
- **Algorithm:** Variational EM
- **Model+Algorithm:** Variational Autoencoders

Nonlinear factor analysis

The linear Gaussian assumption is a very strong constraint on the factor analysis model.

- Essentially, it says the data lie in a **low-dimensional, linear subspace**.
- The assumptions led to tractable EM updates, but is that convenience worth the cost?
- What if the data is low-dimensional but on a **nonlinear manifold**?

Nonlinear factor analysis II

Nonlinear factor analysis relaxes the linearity assumption in order to capture nonlinear relationships between latent variables and data"

$$\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{30}$$

$$\boldsymbol{x}_n \sim \mathcal{N}(f(\boldsymbol{z}_n, \boldsymbol{\theta}), \boldsymbol{\Sigma}) \qquad \text{for } d = 1, \ldots, D. \tag{31}$$

where f is a function mapping the latent variable $\boldsymbol{z}_n$ to expected value of the observation $\boldsymbol{x}_n$, and it is parameterized by $\boldsymbol{\theta}$.

Here we have allowed for arbitrary covariance $\boldsymbol{\Sigma}$, but it's common to constrain it to be diagonal, like in regular factor analysis.

Example

Questions

- How should we parameterize f?
- How can we handle the E-step in nonlinear factor models?

Artificial neural networks as nonlinear function classes for f

We'd like a class of nonlinear functions to map $\boldsymbol{z}_n \to \mathbb{E}[\boldsymbol{x}_n]$.

We've already seen one such mapping in the context of *generalized* linear models (Lap 2).

There, a nonlinear *mean function* was applied to the output of a linear function, $\boldsymbol{w}^\top \boldsymbol{z}_n$.

Standard **artificial neural networks** (ANNs) just stack GLMs one on top of the other.

Each **layer** applies a linear function followed by an elementwise nonlinearity.

$$\mathbb{E}[\boldsymbol{x}_n] = g(\boldsymbol{W}_L g(\boldsymbol{W}_{L-1} g(\dots g(\boldsymbol{W}_1 \boldsymbol{z}_n + \boldsymbol{b}_1)\dots) + \boldsymbol{b}_{L-1}) + \boldsymbol{b}_L) \tag{32}$$

where each $\boldsymbol{W}_\ell \in \mathbb{R}^{P_\ell \times P_{\ell-1}}$, with $P_L = D$ (the data dimension) and $P_0 = P$ (the dimension of $\boldsymbol{z}_n$).

The set of parameters includes $\boldsymbol{\theta} = \{\boldsymbol{W}_\ell, \boldsymbol{b}_\ell\}_{\ell=1}^L$.

Here, g is an elementwise nonlinearity, like a sigmoid, $g(u) = \tanh(u)$, or rectified linear, $g(u) = \max\{u, 0\}$, function.

Lap 6: Factor Analysis, Variational Autoencoders, and Variational EM

- **Model:** Factor Analysis
- **Algorithm:** Expectation Maximization
- **Model:** Nonlinear factor analysis
- **Algorithm: Variational EM**
- **Model+Algorithm:** Variational Autoencoders

The Variational Expectation-Maximization (vEM) algorithm

The **variational** expectation-maximization (vEM) algorithm maximizes the lower bound by iteratively optimizing wrt q and $\boldsymbol{\Theta}$, **subject to the constraint that** $q \in \mathscr{Q}$**.**

1. Initialize parameters $\boldsymbol{\Theta}$
2. Repeat until $\boldsymbol{\Theta}$ (or the ELBO) converges:
   a. **E step:** Set

$$q(\{\boldsymbol{z}_n\}_{n=1}^N; \boldsymbol{\lambda}) = \arg\max_{q \in \mathscr{Q}} \mathscr{L}(q, \boldsymbol{\Theta}) = \arg\min_{q \in \mathscr{Q}} D_{\mathrm{KL}}\left(q(\{\boldsymbol{z}_n\}_{n=1}^N; \boldsymbol{\lambda}) \,\|\, p(\{\boldsymbol{z}_n\}_{n=1}^N \mid \{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta})\right) \quad (33)$$

   b. **M step:** Set

$$\boldsymbol{\Theta} = \arg\max_{\boldsymbol{\Theta}} \mathscr{L}(q, \boldsymbol{\Theta}) \quad (34)$$

$$= \arg\max_{\boldsymbol{\Theta}} \mathbb{E}_q[\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N \mid \boldsymbol{\Theta})] + \log p(\boldsymbol{\Theta}). \quad (35)$$

Question: is the ELBO tight after the E-step? Will $\boldsymbol{\Theta}$ converge to $\arg\max_{\boldsymbol{\Theta}} p(\{\boldsymbol{x}_n\}_{n=1}^N, \boldsymbol{\Theta})$?

The variational E-step for nonlinear factor analysis

For nonlinear factor analysis, we know the true posterior factors over data points since,

$$p(\{\boldsymbol{z}_n\}_{n=1}^N \mid \{\boldsymbol{x}_n\}_{n=1}^N, \Theta) \propto \prod_{n=1}^N \mathcal{N}(\boldsymbol{z}_n \mid \boldsymbol{0}, \boldsymbol{I})\, \mathcal{N}(\boldsymbol{x}_n \mid f(\boldsymbol{z}_n, \boldsymbol{\theta}), \Sigma) \tag{36}$$

$$= \prod_{n=1}^N p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \Theta), \tag{37}$$

where $\Theta = (\boldsymbol{\theta}, \Sigma)$.

We'll assume the variational posterior factors in the same way, $q(\{\boldsymbol{z}_n\}_{n=1}^N; \boldsymbol{\lambda}) = \prod_{n=1}^N q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)$.

How can we solve for the optimal factors though?

$$q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) = \arg\min_{q \in \mathcal{Q}} D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \Theta)\right) \tag{38}$$

Unfortunately the posterior $p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \Theta)$ is no longer a simple closed-form Gaussian.

Stochastic gradient descent on the KL divergence

Even though the true posterior is not a Gaussian, let's approximate it with one. That is, assume,

$$q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) = \mathcal{N}(\boldsymbol{z}_n; \boldsymbol{m}_n, \boldsymbol{C}_n) \tag{39}$$

where $\boldsymbol{\lambda}_n = (\boldsymbol{m}_n, \boldsymbol{C}_n)$.

Then,

$$D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{\Theta})\right) = \mathbb{E}_{q(\boldsymbol{z}_n)}\left[\log q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) - \log p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{\Theta})\right] \tag{40}$$

$$= D_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{z}_n; \boldsymbol{m}_n, \boldsymbol{C}_n) \,\|\, \mathcal{N}(\boldsymbol{z}_n \mid \boldsymbol{0}, \boldsymbol{I})\right) - \mathbb{E}_{q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)}\left[\log p(\boldsymbol{x}_n \mid f(\boldsymbol{z}_n, \boldsymbol{\theta}), \Sigma)\right] + c \tag{41}$$

The first term can be computed in closed form—it's the KL divergence between two Gaussians.

The second term is harder...

The "reparameterization trick"

Note that we can **reparameterize** the variational posterior as

$$\boldsymbol{z}_n \sim \mathcal{N}(\boldsymbol{m}_n, \boldsymbol{C}_n) \qquad \Longleftrightarrow \qquad \boldsymbol{z}_n = \boldsymbol{m}_n + \boldsymbol{C}_n^{1/2}\boldsymbol{\epsilon}_n \tag{42}$$

$$\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{43}$$

We can use the **law of the unconscious statistician** to rewrite the expectation in eq. 41 as,

$$\mathbb{E}_{q(\boldsymbol{z}_n;\boldsymbol{\lambda}_n)}\left[\log p(\boldsymbol{x}_n \mid f(\boldsymbol{z}_n, \boldsymbol{\theta}), \Sigma)\right] = \mathbb{E}_{\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\log p(\boldsymbol{x}_n \mid f(\boldsymbol{m}_n + \boldsymbol{C}_n^{1/2}\boldsymbol{\epsilon}_n, \boldsymbol{\theta}), \Sigma)\right] \tag{44}$$

From this representation, we can rewrite the gradient of the KL divergence as,

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}_n} D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n;\boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \Theta)\right) = \nabla_{\boldsymbol{\lambda}_n} D_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{z}_n; \boldsymbol{m}_n, \boldsymbol{C}_n) \,\|\, \mathcal{N}(\boldsymbol{z}_n \mid \mathbf{0}, \boldsymbol{I})\right) \\ - \mathbb{E}_{\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\nabla_{\boldsymbol{\lambda}_n} \log p(\boldsymbol{x}_n \mid f(\boldsymbol{m}_n + \boldsymbol{C}_n^{1/2}\boldsymbol{\epsilon}_n, \boldsymbol{\theta}), \Sigma)\right]\end{aligned} \tag{45}$$

and **use Monte Carlo to obtain an unbiased estimate of the final expectation.**

The variational M-step for nonlinear factor analysis

The parameters $\boldsymbol{\Theta}$ present the same problem: there's no closed form solution so we need to resort to gradient ascent on the ELBO.

It's slightly easier since q does not depend on $\boldsymbol{\Theta}$, allowing us to bring the gradient inside the expectation,

$$\nabla_{\boldsymbol{\Theta}} \mathscr{L}(q, \boldsymbol{\Theta}) = \nabla_{\boldsymbol{\Theta}} \mathbb{E}_q \left[\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N \mid \boldsymbol{\Theta}) \right] + \nabla_{\boldsymbol{\Theta}} \log p(\boldsymbol{\Theta}) \tag{46}$$

$$= \sum_{n=1}^N \mathbb{E}_{q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)} \left[\nabla_{\boldsymbol{\Theta}} \log \mathcal{N}(\boldsymbol{x}_n \mid f(\boldsymbol{z}_n, \boldsymbol{\Theta}), \Sigma) \right] + \nabla_{\boldsymbol{\Theta}} \log p(\boldsymbol{\Theta}) \tag{47}$$

Again, we can obtain an unbiased Monte Carlo estimate by sampling $q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)$.

What about Σ? You can learn the observation covariance matrix via gradient descent, just like on $\boldsymbol{\Theta}$, but it's common to treat this as a hyperparameter and fix it to $\Sigma = \sigma^2 \boldsymbol{I}$ for some small-ish σ^2.

Then σ^2 effectively reweights the expected log likelihood and the KL to the prior in the ELBO.

Working with mini-batches of data

Note that the ELBO involves a sum over data points,

$$\mathscr{L}(q, \boldsymbol{\Theta}) = \mathbb{E}_q[\log p(\{\boldsymbol{x}_n, \boldsymbol{z}_n\}_{n=1}^N \mid \boldsymbol{\Theta}) - \log q(\{\boldsymbol{z}_n\}_{n=1}^N; \boldsymbol{\lambda}] + \log p(\boldsymbol{\Theta}) \tag{48}$$

$$= \mathbb{E}_{q(\{\boldsymbol{z}_n\}_{n=1}^N; \boldsymbol{\lambda})}\left[\sum_{n=1}^N \log p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \boldsymbol{\Theta}) + \log p(\boldsymbol{z}_n) - \log q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)\right] + \log p(\boldsymbol{\Theta}) \tag{49}$$

$$= \sum_{n=1}^N \Bigg(\underbrace{\mathbb{E}_{q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)}[\log p(\boldsymbol{x}_n \mid \boldsymbol{z}_n)] - D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n)\right)}_{\text{“local ELBO” } \mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\Theta})} \Bigg) + \log p(\boldsymbol{\Theta}) \tag{50}$$

We can view the sum as an “expectation” over data indices,

$$\sum_{n=1}^N \mathscr{L}_n(\lambda_n, \boldsymbol{\Theta}) = N\, \mathbb{E}_{n \sim \mathrm{Unif}(\{1, \ldots, N\})}[\mathscr{L}_n(\lambda_n, \boldsymbol{\Theta})], \tag{51}$$

and we can use Monte Carlo to approximate the expectation and its gradient with respect to $\boldsymbol{\Theta}$.

Lap 6: Factor Analysis, Variational Autoencoders, and Variational EM

- **Model:** Factor Analysis
- **Algorithm:** Expectation Maximization
- **Model:** Nonlinear factor analysis
- **Algorithm:** Variational EM
- **Model+Algorithm: Variational Autoencoders**

Variational Autoencoders (VAEs)

VAEs are essentially nonlinear factor analysis models trained with variational EM, but they introduce **one key idea**.

Note that vEM involves a costly E-step to find the variational parameters $\boldsymbol{\lambda}_n$ for each data point. This could involve many steps of gradient descent inside just the E-step!

With a finite computational budget, we might be better off doing more gradient steps on $\boldsymbol{\Theta}$ and fewer on the local variational parameters.

Note that the optimal variational parameters are just a function of the data point and the model parameters,

$$\boldsymbol{\lambda}_n = \arg\min D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{\Theta})\right) \triangleq g^\star(\boldsymbol{x}_n, \boldsymbol{\Theta}). \tag{52}$$

for some implicit and generally nonlinear function $g^\star$.

Inference networks

VAEs learn an approximation to $g^\star(\boldsymbol{x}_n, \boldsymbol{\Theta})$ with an **inference network** (also called a **recognition network**).

The inference network is (yet another) neural network that takes in a data point $\boldsymbol{x}_n$ and outputs variational parameters $\boldsymbol{z}_n$,

$$\boldsymbol{\lambda}_n \approx g(\boldsymbol{x}_n, \boldsymbol{\phi}), \tag{53}$$

where $\boldsymbol{\phi}$ are the weights of the network.

The advantage is that the inference network is very fast; in the E-step, we simply need to pass a data point through the network to obtain the variational parameters.

The disadvantage is the output will not minimize the KL divergence. However, in practice we might tolerate a worse variational posterior and a weaker lower bound if it buys us more updates of $\boldsymbol{\Theta}$.

Amortization and approximation gaps

Cremer et al. [2018] consider the relative effects of the **amortization gap** and the **approximation gap** on variational EM.

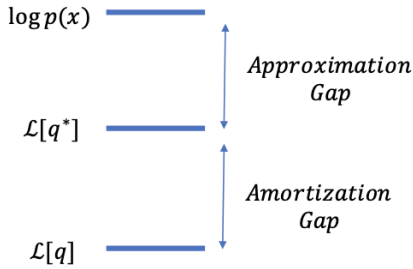
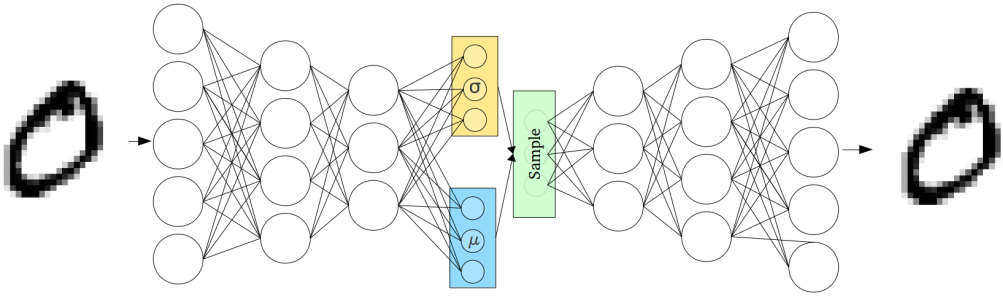


Figure 1. Gaps in Inference

Question: How, if at all, does the generative model $p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \Theta)$ affect these gaps?

VAEs from an autoencoder perspective



From `https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`

Linear VAEs

Question: What does the optimal encoder network look like for linear factor analysis?

References I

Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*, pages 1078–1086. PMLR, 2018.