Variational Autoencoders

STATS305B: Applied Statistics II

Scott Linderman

February 24, 2025

Last Time...

Outline:

- Principal Components Analysis (PCA)
- PCA as a linear Gaussian latent variable model
- Factor analysis
- Linear Dynamical Systems & the Kalman Filter/Smoother

Today...

Outline:

- (Probabilistic) PCA as a (stochastic) linear autoencoder
- Variational Autoencoders (VAEs)
- Review of Gradient-based Variational Inference
- Demo

PCA as a Linear Autoencoder

Last time we introduced PCA as a method for finding dimensions of maximal variance. However, we can arrive at the same model from another perspective: find the linear projection that minimizes the average projection cost.

To formalize this, let

$$\boldsymbol{W} = \begin{bmatrix} | & & | \\ \boldsymbol{w}_1 & \cdots & \boldsymbol{w}_M \\ | & & | \end{bmatrix} \in \mathbb{R}^{D \times M} \quad \text{with} \quad \boldsymbol{W}^\top \boldsymbol{W} = \boldsymbol{I}$$

be an orthogonal basis for the principal subspace.

We will **encode** each data point by subtracting the mean and projecting onto the principal subspace to obtain $\boldsymbol{z}_n = \boldsymbol{W}^\top (\boldsymbol{x}_n - \bar{\boldsymbol{x}})$.

PCA as a Linear Autoencoder

Since $\boldsymbol{W}$ is an orthogonal matrix, all we need to do to **decode** the encoded data point is multiply $\boldsymbol{W}\boldsymbol{z}_n$ and add back the mean. That gives us,

$$\hat{\boldsymbol{x}}_n = \boldsymbol{W}\boldsymbol{z}_n + \bar{\boldsymbol{x}} = \boldsymbol{W}\boldsymbol{W}^\top(\boldsymbol{x}_n - \bar{\boldsymbol{x}}) + \bar{\boldsymbol{x}}.$$

Goal: Find an orthogonal matrix $\boldsymbol{W}$ that *minimizes* the mean squared reconstruction error,

$$\mathscr{L}(\boldsymbol{W}) = \frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n\|_2^2$$

PCA as a Linear Autoencoder

We can write this in matrix notation instead. Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ be the *centered data matrix* with rows $(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^\top$. Then,

$$
\begin{aligned}
\mathscr{L}(\boldsymbol{W}) &= \frac{1}{N} \mathrm{Tr}[(\boldsymbol{X} - \hat{\boldsymbol{X}})^\top (\boldsymbol{X} - \hat{\boldsymbol{X}})] \\
&= \frac{1}{N} \mathrm{Tr}[(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^\top)^\top (\boldsymbol{X} - \boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^\top)] \\
&= \frac{1}{N} \mathrm{Tr}[(\boldsymbol{X}(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top))^\top (\boldsymbol{X}(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top))] \\
&= \frac{1}{N} \mathrm{Tr}[(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)^\top \boldsymbol{X}^\top \boldsymbol{X} (\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)] \\
&= \mathrm{Tr}[(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)^\top \boldsymbol{S} (\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)]
\end{aligned}
$$

where $\boldsymbol{S} = \frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X}$ is the sample covariance matrix.

PCA as a Linear Autoencoder

Now apply the circular trace property,

$$\mathscr{L}(\boldsymbol{W}) = \mathrm{Tr}[\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)^\top]$$

Question: What does $(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)^\top$ equal?

Note that $\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top$ is a projection matrix — it projects a vector onto the *nullspace* of $\boldsymbol{W}$. Applying the projection operator twice doesn't change the result. Mathematically,

$$(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)^\top = \boldsymbol{I} - 2\boldsymbol{W}\boldsymbol{W}^\top + \boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{W}^\top = \boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top$$

where we used the fact that $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{I}$ since $\boldsymbol{W}$ is an orthogonal matrix.

Thus, the objective simplifies to,

$$\mathscr{L}(\boldsymbol{W}) = \mathrm{Tr}[\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{W}\boldsymbol{W}^\top)] = \mathrm{Tr}[\boldsymbol{S}] - \mathrm{Tr}[\boldsymbol{S}\boldsymbol{W}\boldsymbol{W}^\top] = \mathrm{const} - \mathrm{Tr}[\boldsymbol{W}^\top\boldsymbol{S}\boldsymbol{W}].$$

PCA as a Linear Autoencoder

Let $\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ be the eigendecomposition of $\boldsymbol{S}$. (Since it is a covariance matrix, the eigenvectors are orthogonal.) Plugging in,

$$\begin{aligned}
\mathscr{L}(\boldsymbol{W}) &= \text{const} - \operatorname{Tr}[\boldsymbol{W}^\top \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{W}] \\
&= \text{const} - \operatorname{Tr}\left[\boldsymbol{W}^\top \left(\sum_{d=1}^{D} \lambda_d \boldsymbol{u}_d \boldsymbol{u}_d^\top\right) \boldsymbol{W}\right] \\
&= \text{const} - \sum_{m=1}^{M}\sum_{d=1}^{D} \lambda_d \boldsymbol{w}_m^\top \boldsymbol{u}_d \boldsymbol{u}_d^\top \boldsymbol{w}_m \\
&= \text{const} - \sum_{m=1}^{M}\sum_{d=1}^{D} \lambda_d (\boldsymbol{w}_m^\top \boldsymbol{u}_d)^2
\end{aligned}$$

We want to minimize $\mathscr{L}(\boldsymbol{W})$ subject to $\boldsymbol{W}$ being orthogonal. What is the solution? $\boldsymbol{W} = \boldsymbol{U}_M$!

Probabilistic PCA as a Stochastic Linear Autoencoder

The jump from PCA to probabilistic PCA was to treat the scores, $\boldsymbol{z}_n$, as latent variables. We gave them a Gaussian prior and assumed a generative model,

$$\begin{aligned}\boldsymbol{z}_n &\overset{\text{iid}}{\sim} \mathrm{N}(\mathbf{0}, \boldsymbol{I}) \\ \boldsymbol{x}_n \mid \boldsymbol{z}_n &\sim \mathrm{N}(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I}),\end{aligned}$$

where $\boldsymbol{z}_n \in \mathbb{R}^M$ is a latent variable, $\boldsymbol{W} \in \mathbb{R}^{D \times M}$ are the weights, $\boldsymbol{\mu} \in \mathbb{R}^D$ is the bias parameter, and $\sigma^2 \in \mathbb{R}_+$ is a variance.

Given the parameters, we showed that the posterior distribution over latent variables is,

$$p(\boldsymbol{z}_n \mid \boldsymbol{x}_n; \boldsymbol{\theta}) = \mathrm{N}(\boldsymbol{z}_n \mid (\sigma^2 \boldsymbol{I} + \boldsymbol{W}^\top \boldsymbol{W})^{-1} \boldsymbol{W}^\top (\boldsymbol{x}_n - \boldsymbol{\mu}),\ \sigma^2 (\sigma^2 \boldsymbol{I} + \boldsymbol{W}^\top \boldsymbol{W})^{-1}),$$

and when $\sigma^2 \to 0$ and $\boldsymbol{W}$ are the PCs, this converges to a delta function on the PCA scores.

Probabilistic PCA as a Stochastic Linear Autoencoder

To simplify notation, let's rewrite the posterior as,

$$p(\boldsymbol{z}_n \mid \boldsymbol{x}_n; \boldsymbol{\theta}) = \mathrm{N}(\boldsymbol{z}_n \mid \boldsymbol{E}\boldsymbol{x}_n + \boldsymbol{d}, \boldsymbol{G})$$

where

$$\begin{aligned}
\boldsymbol{E} &= (\sigma^2 \boldsymbol{I} + \boldsymbol{W}^\top \boldsymbol{W})^{-1} \boldsymbol{W}^\top \\
\boldsymbol{d} &= -(\sigma^2 \boldsymbol{I} + \boldsymbol{W}^\top \boldsymbol{W})^{-1} \boldsymbol{W}^\top \mu \\
\boldsymbol{G} &= \sigma^2 (\sigma^2 \boldsymbol{I} + \boldsymbol{W}^\top \boldsymbol{W})^{-1}.
\end{aligned}$$

To "autoencode" the data using probabilistic PCA, we could sample $\boldsymbol{z}_n \sim p(\boldsymbol{z}_n \mid \boldsymbol{x}_n; \boldsymbol{\theta})$ by mapping $\boldsymbol{x}_n \mapsto \boldsymbol{E}\boldsymbol{x}_n + \boldsymbol{d}$ and adding Gaussian noise with covariance $\boldsymbol{G}$. Then we can generate a new data point $\hat{\boldsymbol{x}}_n \sim \mathrm{N}(\boldsymbol{W}\boldsymbol{z}_n + \mu, \sigma^2 \boldsymbol{I})$.

Just as PCA could be motivated as finding weights that minimize the reconstruction error, probabilistic PCA can be seen as finding weights that minimize the *expected* reconstruction error (subject to a little regularization).

Revisiting EM for Probabilistic PCA

Recall that we justified EM by connecting it to variational inference. We showed that EM maximizes the evidence lower bound (ELBO),

$$\begin{aligned}\mathscr{L}(q, \boldsymbol{\theta}) &= \sum_n \mathbb{E}_{q(\boldsymbol{z}_n)}\left[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n; \boldsymbol{\theta}) - \log q(\boldsymbol{z}_n)\right] \\ &= \sum_n \underbrace{\mathbb{E}_{q(\boldsymbol{z}_n)}\left[\log p(\boldsymbol{x}_n \mid \boldsymbol{z}_n; \boldsymbol{\theta})\right]}_{\text{expected log likelihood}} - \underbrace{D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n) \,\|\, p(\boldsymbol{z}_n)\right)}_{\text{KL to the prior}} \\ &\leq \sum_n \log p(\boldsymbol{x}_n; \boldsymbol{\theta})\end{aligned}$$

EM can be seen as coordinate ascent on the ELBO:

1. The E-step sets q to the posterior over latent variables, $q(\boldsymbol{z}_n) = p(\boldsymbol{z}_n \mid \boldsymbol{x}_n; \boldsymbol{\theta})$ for the current parameters $\boldsymbol{\theta}$.
2. 2. The M-step updates the parameters to maximize the expected log joint probability.

Revisiting EM for Probabilistic PCA

In probabilistic PCA, maximizing the expected log likelihood is the same as minimizing the expected reconstruction error. Plugging in the definition of the model and the posterior,

$$\begin{aligned}\mathbb{E}_{q(\boldsymbol{z}_n)}\left[\log p(\boldsymbol{x}_n \mid \boldsymbol{z}_n; \boldsymbol{\theta})\right] &= \mathbb{E}_{\mathrm{N}(\boldsymbol{z}_n \mid \boldsymbol{E}\boldsymbol{x}_n + \boldsymbol{d}, \boldsymbol{G})}\left[\log \mathrm{N}(\boldsymbol{x}_n \mid \boldsymbol{W}\boldsymbol{z}_n + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})\right] \\ &= -\frac{1}{2\sigma^2}\mathbb{E}_{\mathrm{N}(\boldsymbol{z}_n \mid \boldsymbol{E}\boldsymbol{x}_n + \boldsymbol{d}, \boldsymbol{G})}\left[\|\boldsymbol{x}_n - \boldsymbol{W}\boldsymbol{z}_n - \boldsymbol{\mu}\|_2^2\right] + \log \sigma + c\end{aligned}$$

where c is an additive constant, and $\boldsymbol{E}$, $\boldsymbol{d}$, $\boldsymbol{G}$ are functions of the parameters, as defined above,

We can't forget about the KL to the prior though! When we maximize the ELBO, we are essentially finding weights that minimize the expected reconstruction error, while also not deviating too far from the standard normal prior on the latent variables.

Variational Autoencoders

Variational Autoencodres (VAEs) are "deep" but conceptually simple generative models. The generative model is the same as probabilistic PCA, but allowing for **nonlinear** mappings between latent variables and observations.

To sample a data point $\boldsymbol{x}_n$,

1. First, sample **latent variables** $\boldsymbol{z}_n$,

$$\boldsymbol{z}_n \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})$$

2. Then sample the data point $\boldsymbol{x}_n$ from a conditional distribution with mean,

$$\mathbb{E}[\boldsymbol{x}_n \mid \boldsymbol{z}_n] = g(\boldsymbol{z}_n; \boldsymbol{\theta}),$$

where $g : \mathbb{R}^H \to \mathbb{R}^D$ is a nonlinear mapping parameterized by $\boldsymbol{\theta}$.

Variational Autoencoders

We will assume g is a simple **feedforward neural network** of the form,

$$g(\boldsymbol{z}; \boldsymbol{\theta}) = g_L(g_{L-1}(\cdots g_1(\boldsymbol{z}) \cdots))$$

where each **layer** is a cascade of a linear mapping followed by an element-wise nonlinearity (except for the last layer, perhaps). For example,

$$g_\ell(\boldsymbol{u}_\ell) = \mathrm{relu}(\boldsymbol{W}_\ell \boldsymbol{u}_\ell + \boldsymbol{b}_\ell); \qquad \mathrm{relu}(a) = \max(0, a).$$

The generative parameters consist of the weights and biases, $\boldsymbol{\theta} = \{\boldsymbol{W}_\ell, \boldsymbol{b}_\ell\}_{\ell=1}^L$.

Learning and Inference

We have two goals. The **learning goal** is to find the parameters that **maximize the marginal likelihood of the data**,

$$\begin{aligned}\boldsymbol{\theta}^\star &= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{X}; \boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \prod_{n=1}^{N} \int p(\boldsymbol{x}_n \mid \boldsymbol{z}_n; \boldsymbol{\theta})\, p(\boldsymbol{z}_n; \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{z}_n\end{aligned}$$

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

$$p(\boldsymbol{z}_n \mid \boldsymbol{x}_n; \boldsymbol{\theta}) = \frac{p(\boldsymbol{x}_n \mid \boldsymbol{z}_n; \boldsymbol{\theta})\, p(\boldsymbol{z}_n; \boldsymbol{\theta})}{\int p(\boldsymbol{x}_n \mid \boldsymbol{z}'_n; \boldsymbol{\theta})\, p(\boldsymbol{z}'_n; \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{z}'_n}$$

Both goals require an integral over $\boldsymbol{z}_n$, but that is intractable for deep generative models.

The Evidence Lower Bound (ELBO)

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

$$
\begin{aligned}
\log p(\boldsymbol{X}; \boldsymbol{\theta}) &= \sum_{n=1}^{N} \log p(\boldsymbol{x}_n; \boldsymbol{\theta}) \\
&\geq \sum_{n=1}^{N} \log p(\boldsymbol{x}_n; \boldsymbol{\theta}) - D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n \mid \boldsymbol{x}_n; \boldsymbol{\theta})\right) \\
&= \sum_{n=1}^{N} \underbrace{\mathbb{E}_{q(\boldsymbol{z}_n)}\left[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n; \boldsymbol{\theta}) - \log q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)\right]}_{\text{"local ELBO"}} \\
&\triangleq \sum_{n=1}^{N} \mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta}) = \mathscr{L}(\boldsymbol{\lambda}, \boldsymbol{\theta})
\end{aligned}
$$

where $\boldsymbol{\lambda} = \{\boldsymbol{\lambda}_n\}_{n=1}^N$. Here, I've written the ELBO as a sum of *local ELBOs* $\mathscr{L}_n$.

Variational Inference

The ELBO is still maximized (and the bound is tight) when each q is equal to the true posterior,

$$q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) = p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}).$$

Unfortunately, the posterior no longer has a simple, closed form.

Question: Suppose $\boldsymbol{x}_n \sim \mathrm{N}(g(\boldsymbol{z}_n; \boldsymbol{\theta}), \boldsymbol{I})$. This deep generative model has a Gaussian prior on $\boldsymbol{z}_n$ and a Gaussian likelihood for $\boldsymbol{x}_n$ given $\boldsymbol{z}_n$. Why isn't the posterior Gaussian?

Nevertheless, we can still constrain q to belong to a simple family. For example, we could constrain it to be Gaussian and seek the best Gaussian approximation to the posterior. This is sometimes called **fixed-form variational inference**. Let,

$$\mathcal{Q} = \left\{ q : q(\boldsymbol{z}; \boldsymbol{\lambda}) = \mathrm{N}\big(\boldsymbol{z} \mid \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \text{ for } \boldsymbol{\lambda} = (\boldsymbol{\mu}, \log \boldsymbol{\sigma}^2) \in \mathbb{R}^{2H} \right\}$$

Variational Inference

Then, for fixed parameters $\boldsymbol{\theta}$, the best q in this **variational family** is,

$$\begin{aligned} q^\star &= \arg\min_{q \in \mathcal{Q}} D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n \mid \boldsymbol{x}_n; \boldsymbol{\theta})\right) \\ &= \arg\max_{\boldsymbol{\lambda}_n \in \mathbb{R}^{2H}} \mathcal{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta}). \end{aligned}$$

Variational Expectation-Maximization (vEM)

Now we can introduce a new algorithm.

Algorithm" Variational EM (vEM) Repeat until either the ELBO or the parameters converges:

1. **M-step:** Set $\boldsymbol{\theta} \leftarrow \arg\max_{\boldsymbol{\theta}} \mathscr{L}(\boldsymbol{\lambda}, \boldsymbol{\theta})$
2. **E-step:** Set $\boldsymbol{\lambda}_n \leftarrow \arg\max_{\boldsymbol{\lambda}_n \in \Lambda} \mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta})$ for $n = 1, \ldots, N$
3. Compute (an estimate of) the ELBO $\mathscr{L}(\boldsymbol{\lambda}, \boldsymbol{\theta})$.

In general, none of these steps will have closed form solutions, so we'll have to use approximations.

Generic M-step with Stochastic Gradient Ascent

For exponential family mixture models, the M-step had a closed form solution. For deep generative models, we need a more general approach.

If the parameters are unconstrained and the ELBO is differentiable wrt $\boldsymbol{\theta}$, we can use stochastic gradient ascent.

$$\begin{aligned}\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \mathscr{L}(q, \boldsymbol{\theta}) \\ &= \boldsymbol{\theta} + \alpha \sum_{n=1}^{N} \mathbb{E}_{q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}_n, \boldsymbol{z}_n; \boldsymbol{\theta}) \right]\end{aligned}$$

Note that the expected gradient wrt $\boldsymbol{\theta}$ can be computed using ordinary Monte Carlo — nothing fancy needed!

The Variational E-step

Assume $\mathcal{Q}$ is the family of Gaussian distributions with diagonal covariance:

$$\mathcal{Q} = \left\{ q : q(\boldsymbol{z}; \boldsymbol{\lambda}) = \mathrm{N}\big(\boldsymbol{z} \mid \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \text{ for } \boldsymbol{\lambda} = (\boldsymbol{\mu}, \log \boldsymbol{\sigma}^2) \in \mathbb{R}^{2H} \right\}$$

This family is indexed by **variational parameters** $\boldsymbol{\lambda}_n = (\boldsymbol{\mu}_n, \log \boldsymbol{\sigma}_n^2) \in \mathbb{R}^{2H}$.

To perform SGD, we need an unbiased estimate of the gradient of the local ELBO, but

$$\begin{aligned} \nabla_{\boldsymbol{\lambda}_n} \mathcal{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta}) &= \nabla_{\boldsymbol{\lambda}_n} \mathbb{E}_{q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)} \left[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n; \boldsymbol{\theta}) - \log q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \right] \\ &\neq \mathbb{E}_{q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)} \left[\nabla_{\boldsymbol{\lambda}_n} \left(\log p(\boldsymbol{x}_n, \boldsymbol{z}_n; \boldsymbol{\theta}) - \log q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \right) \right]. \end{aligned}$$

Reparameterization Trick

One way around this problem is to use the **reparameterization trick**, aka the **pathwise gradient estimator**. Note that,

$$\boldsymbol{z}_n \sim q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \quad \Longleftrightarrow \quad \boldsymbol{z}_n = r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}), \quad \boldsymbol{\epsilon} \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})$$

where $r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}) = \boldsymbol{\mu}_n + \boldsymbol{\sigma}_n \boldsymbol{\epsilon}$ is a reparameterization of $\boldsymbol{z}_n$ in terms of parameters $\boldsymbol{\lambda}_n$ and noise $\boldsymbol{\epsilon}$.

We can use the **law of the unconscious statistician** to rewrite the expectations as,

$$\mathbb{E}_{q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n)} \left[h(\boldsymbol{x}_n, \boldsymbol{z}_n, \boldsymbol{\theta}, \boldsymbol{\lambda}_n) \right] = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathrm{N}(\mathbf{0}, \boldsymbol{I})} \left[h(\boldsymbol{x}_n, r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}), \boldsymbol{\theta}, \boldsymbol{\lambda}_n) \right]$$

where

$$h(\boldsymbol{x}_n, \boldsymbol{z}_n, \boldsymbol{\theta}, \boldsymbol{\lambda}_n) = \log p(\boldsymbol{x}_n, \boldsymbol{z}_n; \boldsymbol{\theta}) - \log q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n).$$

Reparameterization Trick

The distribution that the expectation is taken under no longer depends on the parameters $\boldsymbol{\lambda}_n$, so we can simply take the gradient inside the expectation,

$$\nabla_{\boldsymbol{\lambda}} \mathbb{E}_{q(\boldsymbol{z}_n;\boldsymbol{\lambda}_n)} \left[h(\boldsymbol{x}_n, \boldsymbol{z}_n, \boldsymbol{\theta}, \boldsymbol{\lambda}_n) \right] = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathrm{N}(\mathbf{0}, I)} \left[\nabla_{\boldsymbol{\lambda}_n} h(\boldsymbol{x}_n, r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}), \boldsymbol{\theta}, \boldsymbol{\lambda}_n) \right]$$

Now we can use Monte Carlo to obtain an unbiased estimate of the final expectation!

Working with mini-batches of data

We can view the ELBO as an expectation over data indices,

$$\begin{aligned}\mathscr{L}(\boldsymbol{\lambda}, \boldsymbol{\theta}) &= \sum_{n=1}^{N} \mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta}) \\ &= N\,\mathbb{E}_{n\sim\mathrm{Unif}([N])}[\mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta})].\end{aligned}$$

We can use Monte Carlo to approximate the expectation (and its gradient) by drawing **mini-batches** of data points at random.

In practice, we often cycle through mini-batches of data points deterministically. Each pass over the whole dataset is called an **epoch.**

Algorithm

Now we can add some detail to our variational expectation maximization algorithm.

For epoch $i = 1, \ldots, \infty$:

For $n = 1, \ldots, N$:

1. Sample $\epsilon_n^{(m)} \overset{\text{iid}}{\sim} \mathrm{N}(\mathbf{0}, \boldsymbol{I})$ for $m = 1, \ldots, M$.
2. **M-Step**:

   2.1 Estimate

$$\hat{\nabla}_{\boldsymbol{\theta}} \mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta}) = \frac{1}{M} \sum_{m=1}^{M} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}_n, r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}_n^{(m)}); \boldsymbol{\theta}) \right]$$

   2.2 Set $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_i N \hat{\nabla}_{\boldsymbol{\theta}} \mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta})$

3. **E-step:**

Algorithm

3.1 Estimate

$$\hat{\nabla}_{\lambda}\mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta}) = \frac{1}{M}\sum_{m=1}^{M} \nabla_{\lambda}\Big[\log p(\boldsymbol{x}_n, r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}_n^{(m)}); \boldsymbol{\theta}) - \log q(r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}_n^{(m)}), \boldsymbol{\lambda}_n)\Big]$$

3.2 Set $\boldsymbol{\lambda}_n \leftarrow \boldsymbol{\lambda}_n + \alpha_i \hat{\nabla}_{\boldsymbol{\lambda}}\mathscr{L}_n(\boldsymbol{\lambda}_n, \boldsymbol{\theta})$.

4. Estimate the ELBO

$$\hat{\mathscr{L}}(\boldsymbol{\lambda}, \boldsymbol{\theta}) = \frac{N}{M}\sum_{m=1}^{M} \log p(\boldsymbol{x}_n, r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}_n^{(m)}); \boldsymbol{\theta}) - \log q(r(\boldsymbol{\lambda}_n, \boldsymbol{\epsilon}_n^{(m)}); \boldsymbol{\lambda}_n)$$

5. Decay step size α_i according to schedule.

Amortized Inference

Note that vEM involves optimizing separate variational parameters $\boldsymbol{\lambda}_n$ for each data point. For large datasets where we are optimizing using mini-batches of data points, this leads to a strange asymmetry: we update the generative model parameters $\boldsymbol{\theta}$ every mini-batch, but we only update the variational parameters for the n-th data point once per epoch. Is there any way to share information across data points?

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

$$\boldsymbol{\lambda}_n^\star = \arg\min_{\boldsymbol{\lambda}_n} D_{\mathrm{KL}}\left(q(\boldsymbol{z}_n; \boldsymbol{\lambda}_n) \,\|\, p(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta})\right) \triangleq f^\star(\boldsymbol{x}_n, \boldsymbol{\theta}).$$

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

VAEs learn an approximation to $f^\star(\boldsymbol{x}_n, \boldsymbol{\theta})$ with an **inference network**, a.k.a. **recognition network** or **encoder**.

Amortized Inference

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

$$\boldsymbol{\lambda}_n \approx f(\boldsymbol{x}_n, \boldsymbol{\phi}),$$

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

The advantage is that the inference network shares information across data points — it *amortizes* the cost of inference, hence the name. 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 leads to faster optimization of the ELBO overall.

Putting it all together

Logically, I find it helpful to distinguish between the E and M steps, but with recognition networks and stochastic gradient ascent, the line is blurred.

The final algorithm looks like this.

Variational EM (with amortized inference)

Repeat until either the ELBO or the parameters converges:

1. Sample data point $n \sim \text{Unif}(1, \dots, N)$. [Or a minibatch of data points.]
2. Estimate the local ELBO $\mathcal{L}_n(\boldsymbol{\phi}, \boldsymbol{\theta})$ with Monte Carlo. [Note: it is a function of $\boldsymbol{\phi}$ instead of $\boldsymbol{\lambda}_n$.]
3. Compute unbiased Monte Carlo estimates of the gradients $\widehat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_n(\boldsymbol{\phi}, \boldsymbol{\theta})$ and $\widehat{\nabla}_{\boldsymbol{\phi}} \mathcal{L}_n(\boldsymbol{\phi}, \boldsymbol{\theta})$. [The latter requires the reparameterization trick.]

Putting it all together

4. Set

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_i \widehat{\nabla}_{\boldsymbol{\theta}} \mathscr{L}_n(\boldsymbol{\phi}, \boldsymbol{\theta})$$
$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha_i \widehat{\nabla}_{\boldsymbol{\phi}} \mathscr{L}_n(\boldsymbol{\phi}, \boldsymbol{\theta})$$

with step size α_i decreasing over iterations i according to a valid schedule.