

Variational Inference:

Auto-Encoding Variational Bayes

By: Gursimran Singh & Wilder Lavington & Michael Przystupa

Department of Computer Science,
University of British Columbia, Vancouver BC

- Goal: Approximate the true posterior of a directed graphical model despite computation intractability
- Mean field methods
 - Require analytic solutions of expectation w.r.t to parameters
 - Computationally intractable in general case
- Proposed solution: Reparameterize variational lower bound
 - On IID data allows efficient inference for big data sets
 - Applications in recognition, denoising, representation, and visualization

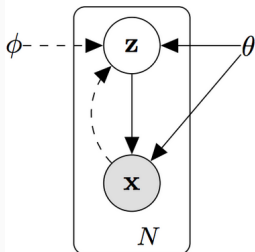
¹[5]

Problem Scenario: Set-up

- We're given IID data $\{x_0, x_1, \dots, x_n\}$
- Assume following Graphical Model: $P_\theta(x, z) = P_\theta(z)P_\theta(x|z)$
- Goals for our algorithm:
 1. Works when marginalizing intractable R.V.
 - e.g. $P(x) = \int p(z)p(x|z)dz$ is hard
 2. Efficient on approximating posterior distribution for large datasets
 - Being able to do mini-batch updates would be nice
- Related Problems:
 1. Efficient approximate ML/MAP estimation for parameters θ
 - θ could represent a simulation of some process
 2. Want efficient posterior inference of z : $P(z|x) = \frac{P_\theta(z)P_\theta(x|z)}{P_\theta(x)}$
 - Useful for representing data (e.g. language embeddings)
 3. Efficiently approximate the marginal inference of the variable x
 - Applications: image denoising, inpainting, and super-resolution

Problem Scenario: Proposed Solution

- Probabilistic Encoder ²
 - $q_{\phi}(z|x) \approx p_{\theta}(z|x)$
 - the recognition model
 - variables can be dependent and computed in non-closed form
- Probabilistic Decoder ³
 - $p_{\theta}(x|z)$
 - the generative model
- Optimize both $q_{\phi}(z|x)$ and $p_{\theta}(z|x)$ jointly



²dashed lines in diagram

³solid lines in diagram

The Variational Bound

We assumed our data was IID, so the marginal likelihood of the joint distribution over our data is:

$$\log p_{\theta}(x^{(1)}, x^{(2)}, \dots, x^{(N)}) = \sum_{i=1}^N \log p_{\theta}(x^{(i)})$$

Where $\log p_{\theta}(x^{(i)})$ can be derived as follows ⁴:

$$\begin{aligned} \log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z | x^{(i)})} \left[\log p_{\theta}(x^{(i)}) \right] && (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] && (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \frac{q_{\phi}(z | x^{(i)})}{q_{\phi}(z | x^{(i)})} \right] && (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[\log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] && (\text{Logarithms}) \\ &= \underbrace{\mathbf{E}_z \left[\log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)})) \end{aligned}$$

⁴borrowed from <https://www.cs.ubc.ca/~lsigal/532L/Lecture11.pdf>

The Variational Bound Continued

Given our derivation of $\log p_{\theta}(x^{(i)})$:

$$\log p_{\theta}(x^{(i)}) = D_{KL}(q_{\phi}(z|x^{(i)})||p_{\theta}(z|x^{(i)})) + L(\theta, \phi; x^{(i)})$$

We can rewrite this as a lower bound by ignoring D_{KL} because $D_{KL} \geq 0$

$$\log p_{\theta}(x^{(i)}) \geq L(\theta, \phi; x^{(i)})$$

$L(\theta, \phi; x^{(i)})$ is referred to as the variational lower bound and is our objective which we'd like to optimize w.r.t ∇_{ϕ} & ∇_{θ}

Whats the issue?

Using a naive Monte-Carlo gradient estimator, the gradient w.r.t ϕ can have extremely high variance, and can effect the algorithms ability to optimize efficiently. This can be seen by considering:

$$\nabla_{\phi} E_{q_{\phi}(z)}[f(z)] = E_{q_{\phi}(z)}[f(z) \nabla_{\phi} \log q_{\phi}(z)] \approx \frac{1}{L} \sum_{l=1}^L f(z) \nabla_{\phi} \log q_{\phi}(z^{(l)})$$

Where $z^{(l)} \sim q_{\phi}(z|x^{(i)})$

REINFORCE Trick

As a quick aside, the approximation giving above follows from the classic REINFORCE trick. A nice derivation by Tuan Anh Le [7] is given below:

Consider a random variable $X : \Omega \rightarrow \mathcal{X}$ whose distribution is parameterized by ϕ ; and a function $f : \mathcal{X} \rightarrow \mathbb{R}$. The goal is to approximate $\frac{\partial}{\partial \phi} \mathbb{E}[f(X)]$.

Let $p_\phi(x)$ be the density of X with respect to the base measure $\mathrm{d}x$. Using the identity $\frac{\partial}{\partial \phi} p_\phi(x) = p_\phi(x) \frac{\partial}{\partial \phi} \log p_\phi(x)$, we get:

$$\frac{\partial}{\partial \phi} \mathbb{E}[f(X)] = \frac{\partial}{\partial \phi} \int_{\mathcal{X}} f(x) p_\phi(x) \mathrm{d}x \quad (1)$$

$$= \int_{\mathcal{X}} f(x) \frac{\partial}{\partial \phi} p_\phi(x) \mathrm{d}x \quad (2)$$

$$= \int_{\mathcal{X}} f(x) \frac{\partial}{\partial \phi} \log p_\phi(x) p_\phi(x) \mathrm{d}x \quad (3)$$

$$= \mathbb{E} \left[f(x) \frac{\partial}{\partial \phi} \log p_\phi(x) \right]. \quad (4)$$

Hence, $\frac{\partial}{\partial \phi} \mathbb{E}[f(X)]$ can be approximated by a Monte Carlo estimator:

$$\frac{1}{N} \sum_{n=1}^N f(X^n) \frac{\partial}{\partial \phi} \log p_\phi(X^n) \quad X_n \sim p_\phi, n = 1, \dots, N. \quad (5)$$

Thus, we only need $\log p_\phi(x)$ to be differentiable with respect to ϕ . This estimator applicable to a wide range of distributions of X but suffers from high variance (why?).

Random Variable Transformation

For a chosen approximate posterior $q_\phi(z|x)$ we can reparameterize the random variable distributed by the posterior such that,

$$\hat{z} \sim q_\phi(z|x)$$

using a differentiable transformation $g_\phi(\epsilon, x)$ with an auxiliary noise variable $\epsilon \sim p(\epsilon)$ we then have

$$\hat{z} \sim g_\phi(\epsilon, x) \text{ with } \epsilon \sim p(\epsilon)$$

Where $g_\phi(\epsilon, x)$ is some vector valued function parameterized by ϕ , and $p(\epsilon)$ is some independent marginal pdf.

Reparameterization Trick: Explanation

Why is it useful?

Reparameterization is used so that we can rewrite the expectation w.r.t. $q_\phi(z|x)$ such that the monte-carlo estimate of the expectation is differentiable w.r.t. ϕ .

This is because for some deterministic mapping $z = g_\epsilon(\epsilon, x)$ we know:

$$q_\phi(z|x) \prod_i dz_i = p(\epsilon) \prod_i d\epsilon_i$$

Which means,

$$\int q_\phi(z|x) f(z) dz = \int p(\epsilon) f(z) d\epsilon = \int p(\epsilon) f(g_\epsilon(\epsilon, x)) d\epsilon$$

This means we can now construct a differential operator

$$\int q_\phi(z|x) f(z) dz \approx \frac{1}{L} \sum_{l=1}^L f(g_\phi(x, \epsilon^l))$$

which can conveniently be used for the estimator of the lower bound.

Reparameterization Trick: Example

Consider the Univariate Gaussian case where $z \sim p(z|x) = N(\mu, \sigma^2)$. A valid reparameterization would be $z = \mu + \sigma\epsilon$ where $\epsilon \sim N(0, 1)$. This means that:

$$E_{N(z; \mu, \sigma^2)}[f(z)] = E_{N(\epsilon; 0, 1)}[f(\mu + \sigma\epsilon)] \approx \frac{1}{L} \sum_{l=1}^L f(\mu + \sigma\epsilon_l)$$

How to pick a reparameterization:

Which variational approximations $q_{\phi}(z|x)$ actually have such a transformation g , and how do we know what to pick for g ? The paper recommends four main approaches:

- Tractable Inverse CDF: Let $\epsilon \sim U(0, 1)$ and let $g_{\theta}(\epsilon, x)$ be the inverse cdf of $q_{\phi}(z|x)$.
- Location scale families: for any location scale family (i.e. a distribution that is parameterized by some notion of position and scaling), we can choose a standard distribution as the auxiliary variable and let $g = \text{location} + \text{scale} * \epsilon$.
- Composition: Can we express the random variable as some specific transformation of an auxiliary random variable (i.e. logNormal ect.)
- otherwise use more expensive CDF inverse methods that have time complexity.

Monte-Carlo Estimates using Transformation

We can now set up Monte-Carlo Estimates of the expectations of some function $f(z)$ w.r.t. $q_\phi(z|x)$:

$$E_{q_\phi(z)}[f(z)] = E_{p(\epsilon)}[f(g_\phi(\epsilon, x^{(i)}))] \approx \sum_{l=1}^L f(g_\phi(\epsilon^{(l)}, x^{(i)}))$$

Where $\epsilon^{(l)}$ are distributed according to p . We can then again apply our variational lower bound to get what is defined in the paper as Stochastic Gradient Variational Bayes (SGVB) estimator. This can be written more explicitly as,

$$L(\theta, \phi, x^{(i)}) \approx L^A(\theta, \phi, x^{(i)}) = \frac{1}{L} \sum_{l=1}^L \log(p_\theta(x^{(i)}, z^{(i,l)}) - \log(q_\phi(z^{(i,l)}|x^{(i)}))$$

Where $z^{(i,l)} = g_\phi(\epsilon^{(i,l)}, x^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$

A Special Case of the algorithm:

Its often the case that $KL(q_\phi(z|x^{(i)})||p_\theta(z))$ is analytically integrable, such that only the expected reconstruction error ($p_\theta(x^{(i)}|z)$) requires estimation via sampling.

In this case the KL divergence term "regularizes" ϕ and encourages the approximate posterior to be close to the prior. This alternative lowerbound estimator is given as:

$$L(\theta, \phi, x^{(i)}) \approx L^B(\theta, \phi, x^{(i)}) = -KL(q_\phi(z|x^{(i)})||p_\theta(z)) + \frac{1}{L} \sum_{l=1}^L \log(p_\theta(x^{(i)}|z^{(i,l)}))$$

Where $z^{(i,l)} = g_\phi(\epsilon^{(i,l)}, x^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$. This formulation when applicable generally has lower variance then the generic approach.

Mini-batch implementation: Explanation

Given multiple data points from some data set X with N data points, we can construct an estimator of the marginal likelihood lowerbound of the full dataset using mini batches:

$$L(\theta, \phi, X) \approx L^M(\theta, \phi, X^M) = \frac{N}{M} \sum_{i=1}^M L(\theta, \phi, x^{(i)})$$

Where $L(\theta, \phi, x^{(i)})$ is approximated in one of the two ways discussed above and our mini batch X^M is a set of m points drawn at random from the total data set X of size N .

Mini-batch implementation: Algorithm

Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\theta, \phi \leftarrow$ Initialize parameters

repeat

$\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)

$\epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$

$\mathbf{g} \leftarrow \nabla_{\theta, \phi} \tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M, \epsilon)$ (Gradients of minibatch estimator (8))

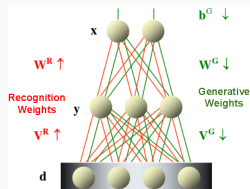
$\theta, \phi \leftarrow$ Update parameters using gradients \mathbf{g} (e.g. SGD or Adagrad [DHS10])

until convergence of parameters (θ, ϕ)

return θ, ϕ

Related Work : Helmholtz Machine⁵

- Precursor to Auto-encoding Variational Bayes
 - 1995 by Dayan et. al
 - Learns posterior distribution of $p_{\theta}(x)$ (G in next few slides)
 - Designed for use on bit vectors
- Activations in network are sigmoid function σ
 - Treat each layer in neural network as random variable
 - We sample from each layer: $z_l \sim \sigma(W_{l-1}z_{l-1})$
- Optimize Helmholtz free energy with wake-sleep algorithm
 - $$\underbrace{F_G(x)}_{\text{free energy}} = \underbrace{E_G[-\log p_{\theta}(x, z)]}_{\text{internal "Energy" }} - \underbrace{TH_G(z|d)}_{\text{Temp * Entropy}}$$



⁵[1, 6]

Helmholtz Machine: Objective

- Hard to take gradients with respect to $F_g(x)$
- Introduce $p_\phi(z|d)$ (refer to as R) & is used to minimize $KL[p_R(z|d), p_G(z|d)]$
 - Want R & G to learn from each other to approximate true posterior
- With some derivation this gives variational free energy
 - $F_G^R \equiv E_R[-\log p(z, x)] - H_R(z|d) = F_G(x) + KL[p_R(z|d), p_G(z|d)]$
- Wake Sleep Algorithm alternates between updating R & G

```
 $\mathbf{V}^G, \mathbf{W}^G, \mathbf{b}^G = \mathbf{0}$   
 $\mathbf{V}^R, \mathbf{W}^R = \mathbf{0}$   
repeat  
  wake phase to change  $\mathbf{V}^G, \mathbf{W}^G, \mathbf{b}^G$   
  sleep phase to change  $\mathbf{V}^R, \mathbf{W}^R$   
until  
   $KL[p(\mathbf{D}), p_G(\mathbf{D})]$  is sufficiently close to 0.
```

Helmholtz Machine: Wake-Sleep Details

- Wake Phase: Updates parameters of Generative network
 - $F_G(x) + KL[p_R(z|d), p_G(z|d)]$
- Sleep Phase: Updates parameters of Recognition network
 - $F_G(x) + KL[p_G(z|d), p_R(z|d)]$ ⁶

WAKE PHASE

// Experience reality!

d = getSampleFromWorld()

// Pass sense datum up through recognition network

y = SAMPLE($\sigma(\mathbf{V}^R [\mathbf{d} \parallel 1]^T)$)

x = SAMPLE($\sigma(\mathbf{W}^R [\mathbf{y} \parallel 1]^T)$)

// Pass back down through generation network, saving computed probabilities

$\xi = \sigma(\mathbf{b}^G)$

$\psi = \sigma(\mathbf{W}^G [\mathbf{x} \parallel 1]^T)$

$\delta = \sigma(\mathbf{V}^G [\mathbf{y} \parallel 1]^T)$

// Adjust generative weights by delta rule

$\mathbf{b}^G \ += \ \varepsilon (\mathbf{x} - \xi)$

$\mathbf{W}^G \ += \ \varepsilon (\mathbf{y} - \psi) [\mathbf{x} \parallel 1]^T$

$\mathbf{V}^G \ += \ \varepsilon (\mathbf{d} - \delta) [\mathbf{y} \parallel 1]^T$

SLEEP PHASE

// Initiate a dream !

x = SAMPLE($\sigma(\mathbf{b}^G)$)

// Pass dream signal down through generation network

y = SAMPLE($\sigma(\mathbf{W}^G [\mathbf{x} \parallel 1]^T)$)

d = SAMPLE($\sigma(\mathbf{V}^G [\mathbf{y} \parallel 1]^T)$)

// Pass back up through recognition network, saving computed probabilities

$\psi = \sigma(\mathbf{V}^R [\mathbf{d} \parallel 1]^T)$

$\xi = \sigma(\mathbf{W}^R [\mathbf{y} \parallel 1]^T)$

// Adjust recognition weights by delta rule

$\mathbf{V}^R \ += \ \varepsilon (\mathbf{y} - \psi) [\mathbf{d} \parallel 1]^T$

$\mathbf{W}^R \ += \ \varepsilon (\mathbf{x} - \xi) [\mathbf{y} \parallel 1]^T$

⁶What's the issue with doing this?

Related Work: Stochastic Backpropagation and Approximate Inference in Deep Generative Models ⁷

The generative process within deep latent Gaussian models (DLGMs) proceeds as follows:

- Model consists of L layers of latent variables
- To generate a sample, we draw the distribution at the top most layer
- The activation at any layer below is formed by a nonlinear activation function perturbed by Gaussian noise.
- Observations are then generated by descending through this hierarchy, and sampling from the activation at the lowest layer.

More succinctly:

$$\boldsymbol{\xi}_l \sim \mathcal{N}(\boldsymbol{\xi}_l | \mathbf{0}, \mathbf{I}), \quad l = 1, \dots, L$$

$$\mathbf{h}_L = \mathbf{G}_L \boldsymbol{\xi}_L,$$

$$\mathbf{h}_l = T_l(\mathbf{h}_{l+1}) + \mathbf{G}_l \boldsymbol{\xi}_l, \quad l = 1 \dots L - 1$$

$$\mathbf{v} \sim \pi(\mathbf{v} | T_0(\mathbf{h}_1)),$$

ε are mutually independent Gaussians,

T_l are multilayer perceptrons (MLPs),

G_l are matrices,

$\pi(\mathbf{v})$ is a distribution parameterized by the first latent layer.

⁷[9]

Variational Inference with Deep Latent Gaussian Models

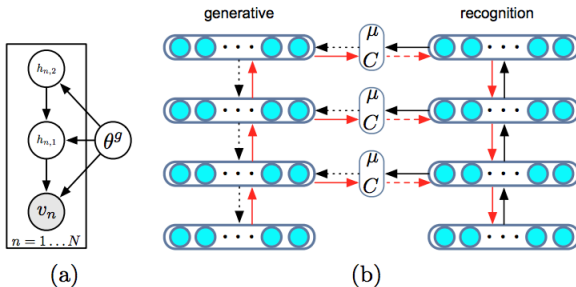


Figure 1. (a) Graphical model for DLGMs (5). (b) The corresponding computational graph. Black arrows indicate the forward pass of sampling from the recognition and generative models: Solid lines indicate propagation of deterministic activations, dotted lines indicate propagation of samples. Red arrows indicate the backward pass for gradient computation: Solid lines indicate paths where deterministic backpropagation is used, dashed arrows indicate stochastic backpropagation.

Scalable Inference in DLGMs

The general optimization algorithm w.r.t the parameters using mini-batch:

Algorithm 1 Learning in DLGMs

```
while hasNotConverged() do  
   $\mathbf{V} \leftarrow \text{getMiniBatch}()$   
   $\boldsymbol{\xi}_n \sim q(\boldsymbol{\xi}_n | \mathbf{v}_n)$  (bottom-up pass) eq. (12)  
   $\mathbf{h} \leftarrow \mathbf{h}(\boldsymbol{\xi})$  (top-down pass) eq. (3)  
  updateGradients() eqs (14) – (17)  
   $\boldsymbol{\theta}^{g,r} \leftarrow \boldsymbol{\theta}^{g,r} + \Delta \boldsymbol{\theta}^{g,r}$   
end while
```

$$\begin{aligned} \mathcal{F}(\mathbf{V}) = & - \sum_n \mathbb{E}_q [\log p(\mathbf{v}_n | \mathbf{h}(\boldsymbol{\xi}_n))] + \frac{1}{2\kappa} \|\boldsymbol{\theta}^g\|^2 \\ & + \frac{1}{2} \sum_{n,l} [\|\boldsymbol{\mu}_{n,l}\|^2 + \text{Tr}(\mathbf{C}_{n,l}) - \log |\mathbf{C}_{n,l}| - 1], \end{aligned}$$

$$\nabla_{\boldsymbol{\theta}^r} \mathcal{F}(\mathbf{v}) = \nabla_{\boldsymbol{\mu}} \mathcal{F}(\mathbf{v})^\top \frac{\partial \boldsymbol{\mu}}{\partial \boldsymbol{\theta}^r} + \text{Tr} \left(\nabla_{\mathbf{R}} \mathcal{F}(\mathbf{v}) \frac{\partial \mathbf{R}}{\partial \boldsymbol{\theta}^r} \right).$$

$$\Delta \boldsymbol{\theta}^{g,r} = -\Gamma^{g,r} \nabla_{\boldsymbol{\theta}^{g,r}} \mathcal{F}(\mathbf{V}),$$

Where $\Gamma^{g,r}$ is a preconditioning matrix, and \mathbf{R} is the factor matrix.

Gaussian Covariance Parameterization

We can either pick a diagonal covariance, or improve on the diagonal approximation by parameterizing as a rank-1 matrix with diagonal correction.

- If we define the precision matrix:

$$\mathbf{C}^{-1} = \mathbf{D} + \mathbf{u}\mathbf{u}^\top.$$

- Apply an matrix inversion lemma:

$$\mathbf{C} = \mathbf{D}^{-1} - \eta \mathbf{D}^{-1} \mathbf{u} \mathbf{u}^\top \mathbf{D}^{-1},$$

$$\log |\mathbf{C}| = \log \eta - \log |\mathbf{D}|.$$

- and calculate $\log |\mathbf{C}|$:

$$\log |\mathbf{C}| = \log \eta - \log |\mathbf{D}|.$$

- Compute \mathbf{R} using:

$$\mathbf{R} = \mathbf{D}^{-\frac{1}{2}} - \left[\frac{1 - \sqrt{\eta}}{\mathbf{u}^\top \mathbf{D}^{-1} \mathbf{u}} \right] \mathbf{D}^{-1} \mathbf{u} \mathbf{u}^\top \mathbf{D}^{-\frac{1}{2}}.$$

- The product of a vector with \mathbf{R} cost $O(k)$ without computing \mathbf{R} explicitly.
- So we can efficiently sample from the Gaussian model, $\xi = \mu + \mathbf{R}\epsilon$

Which now allows us to compute the Gaussian KL in $O(K)$ operations, the covariance scales linearly in latent variables, and the variational distribution can be parameterized jointly.

Variational Autoencoder: Recap

General estimator: The reparametrized SGVB estimator

$$L(\theta, \phi, x^{(i)}) \approx L^A(\theta, \phi, x^{(i)}) = \frac{1}{L} \sum_{l=1}^L \log(p_{\theta}(x^{(i)}, z^{(i,l)}) - \log(q_{\phi}(z^{(i,l)}|x^{(i)}))$$

Where $z^{(i,l)} = g_{\phi}(\epsilon^{(i,l)}, x^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$

This contains expectations over both terms

Simplification: For some specific priors p_{θ} and approximate variational posterior $q_{\phi}(z|x)$, **integrate the KL** term analytically.

New estimator: Contains **expectation in only one term**.

$$L(\theta, \phi, x^{(i)}) \approx L^B(\theta, \phi, x^{(i)}) = -D_{KL}(q_{\phi}(z|x^{(i)})||p_{\theta}(z)) + \frac{1}{L} \sum_{l=1}^L \log(p_{\theta}(x^{(i)}|z^{(i,l)}))$$

Where $z^{(i,l)} = g_{\phi}(\epsilon^{(i,l)}, x^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$

Variational Autoencoder: distributions

Prior -> Isotropic Gaussian distribution

$$p_{\theta}(z) = \mathcal{N}(z; 0, \mathcal{I})$$

Approximate posterior -> Gaussian (diagonal covariance)

$$q_{\phi}(z|x^i) = \mathcal{N}(z; \mu^i, \sigma^{2i}\mathcal{I})$$

- μ^i and σ^i are vectors of size latent space (J)
- μ^i and σ^i are computed using **neural network** input on x .

Likelihood -> Multivariate (diagonal covariance)

$$p_{\theta}(x|z) = \mathcal{N}(x; \mu, \sigma^2\mathcal{I})$$

$$\text{where } \mu = W_4 h + b_4; \sigma^2 = W_5 h + b_5; h = \tanh(W_3 z + b_3)$$

- Real valued outputs -> Gaussian distribution
- Binary outputs -> Bernoulli

Varitational Autoencoder - Objective

Substitute

$$p_{\theta}(z) = \mathcal{N}(z; 0, \mathcal{I}); q_{\phi}(z|x^i) = \mathcal{N}(z; \mu^i, \sigma^{2i}\mathcal{I})$$

Integrate the KL term

$$-D_{KL}((q_{\phi}(z|x^i)||p_{\theta}(z)) = \int q_{\phi}(z|x^i)(\log p_{\theta}(z) - \log q_{\theta}(z|x^i))dz$$

VAE estimator

$$L(\theta, \phi; x^{(i)}) \approx \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^{(i)})^2 - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) + \frac{1}{L} \sum_{l=1}^L \log p_{\theta}(x^{(i)}|z^{(i,l)})$$

Where $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)}\epsilon^{(l)}$ and $\epsilon \sim N(0, I)$.

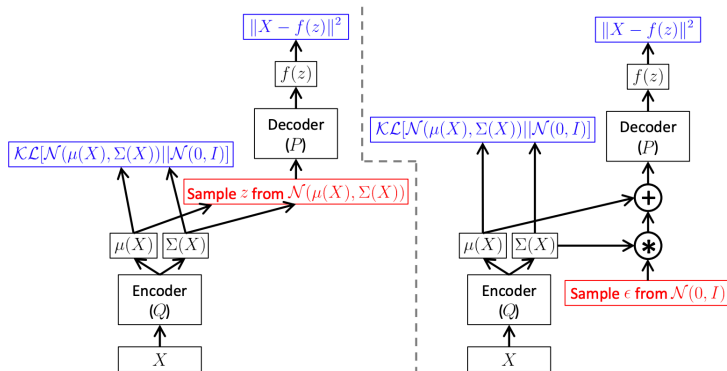
We can optimize this using the AEVB algorithm

$$\begin{aligned}\int q_{\theta}(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}) d\mathbf{z} \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (\mu_j^2 + \sigma_j^2)\end{aligned}$$

$$\begin{aligned}\int q_{\theta}(\mathbf{z}) \log q_{\theta}(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) d\mathbf{z} \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (1 + \log \sigma_j^2)\end{aligned}$$

$$\begin{aligned}-D_{KL}((q_{\phi}(\mathbf{z})||p_{\theta}(\mathbf{z}))) &= \int q_{\theta}(\mathbf{z}) (\log p_{\theta}(\mathbf{z}) - \log q_{\theta}(\mathbf{z})) d\mathbf{z} \\ &= \frac{1}{2} \sum_{j=1}^J (1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2)\end{aligned}$$

VAE - as a neural network [2]



Left - Stochasticity exists between the encoder and the decoder
Right (reparameterized) - No stochasticity between the encoder and the decoder

Interpreting the objective

Reconstruction loss

$$E_{q_{\phi}(z|x)}[\log p_{\theta}(x|z)] = \frac{1}{L} \sum_{l=1}^L \log p_{\theta}(x^{(i)}|z^{(i,l)})$$

- Encourages the decoder to reconstruct the data
- Same as MSE if we use a Gaussian likelihood
- Sampling encourages smoothness in the latent space
- Without sampling this corresponds to standard auto-encoder

Regularization

$$-D_{KL}((q_{\phi}(z|x^i)||p_{\theta}(z)) = \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^{(i)})^2 - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2)$$

- Encourages the $q_{\phi}(z|x)$ to get close to $p_{\theta}(z) \sim \mathcal{N}(0, \mathcal{I})$
- Act as a regularization term over the standard auto-encoder
- Forces the autoencoder, to learn useful representations rather than merely reproducing data

Application: Representation learning⁸

Apart from generation, VAEs are expected to learn **informative codes**.

VAE Objective : $L(\theta, \phi; x^i) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathbb{D}_{\text{KL}}(q_\phi(z|x) || p_\theta(z))$

Ignored codes problem

- The KL term encourages $q_\phi(z|x)$ to be same as $\mathcal{N}(0, \mathcal{I})$
- If $p_\theta(x|z)$ is too expressive, z can be $\mathcal{N}(0, \mathcal{I})$ for all x
- Mutual information between x and z vanishes
- This is known as **information preference problem**

Entangled codes problem

- The likelihood term encourages accurate reconstruction.
- It forces z to be distinct, informative and separated
- This may lead to entangled and inefficient codes

⁸burgess2018understanding

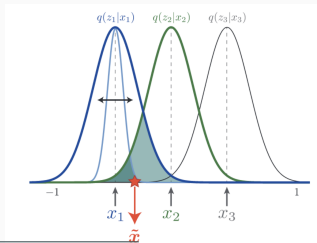
Entangled codes problem (beta-VAE)⁹ part 1

Solution: Increase the pressure on the KL term (hyperparameter)[3]

$$L(\theta, \phi; x^i) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta \mathbb{D}_{\text{KL}}(q_\phi(z|x) || p_\theta(z))$$

Encourages smoothness

- Close values in z space, leads to smooth changes in x
- Since $q_\phi(z_l^i|x^i)$ can overlap, $p(x|x)$ should to still make sense



⁹burgess2018understanding

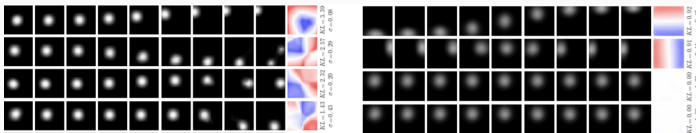
Entangled codes problem (beta-VAE) part 2

Encourages parsimony

- Compress information about z into fewer dimensions
- Less important factors of variation can be $\mathcal{N}(0, 1)$

Encourages axis-alignment

- Aligns the main factors of variation with the dimensions of z
- Different variances for dimensions only possible if aligned



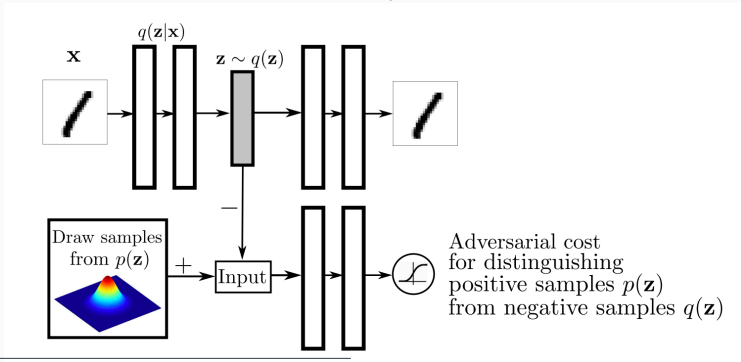
Information preference problem (info-VAE) ¹⁰

Solution: Divergence which increases the x and z mutual information

$$\mathcal{L}_{\text{InfoVAE}}(\theta, \phi; x) = \mathbb{E}_{q_{\phi}(z|x)}[\log p_{\theta}(x|z)] - (1 - \alpha)\mathbb{D}_{\text{KL}}(q_{\phi}(z|x)||p_{\theta}(z)) - (\alpha + \lambda - 1)\mathbb{D}_{\text{KL}}(q_{\phi}(z)||p_{\theta}(z))$$

Adversarial auto-encoders[8] $\rightarrow \alpha = 1$ and $\lambda = 1$

Matches the aggregated posterior $q_{\phi}(z)$ with the prior $p_{\theta}(z)$



Motivation: Make the variational approximation bound tighter

Standard VAE objective

$$\begin{aligned}\nabla_{\theta} \log \mathbb{E}_{\mathbf{h} \sim q(\mathbf{h}|\mathbf{x}, \theta)} \left[\frac{p(\mathbf{x}, \mathbf{h}|\theta)}{q(\mathbf{h}|\mathbf{x}, \theta)} \right] &= \nabla_{\theta} \mathbb{E}_{\epsilon^1, \dots, \epsilon^L \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\log \frac{p(\mathbf{x}, \mathbf{h}(\epsilon, \mathbf{x}, \theta)|\theta)}{q(\mathbf{h}(\epsilon, \mathbf{x}, \theta)|\mathbf{x}, \theta)} \right] \\ &= \mathbb{E}_{\epsilon^1, \dots, \epsilon^L \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\nabla_{\theta} \log \frac{p(\mathbf{x}, \mathbf{h}(\epsilon, \mathbf{x}, \theta)|\theta)}{q(\mathbf{h}(\epsilon, \mathbf{x}, \theta)|\mathbf{x}, \theta)} \right] \\ &= \frac{1}{k} \sum_{i=1}^k \nabla_{\theta} \log w(\mathbf{x}, \mathbf{h}(\epsilon_i, \mathbf{x}, \theta), \theta)\end{aligned}$$

¹¹burda2015importance

Importance weighted autoencoder: part1

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}_{\mathbf{h}_1, \dots, \mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})} \left[\log \frac{1}{k} \sum_{i=1}^k \frac{p(\mathbf{x}, \mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})} \right]$$

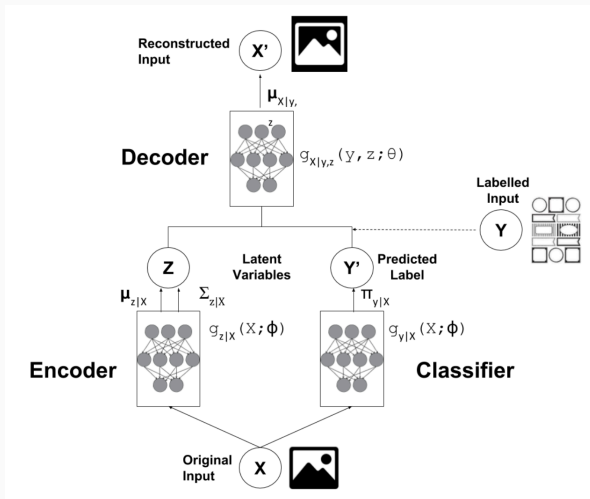
$$\begin{aligned} \nabla_{\theta} \mathcal{L}_k(\mathbf{x}) &= \nabla_{\theta} \mathbb{E}_{\mathbf{h}_1, \dots, \mathbf{h}_k} \left[\log \frac{1}{k} \sum_{i=1}^k w_i \right] = \nabla_{\theta} \mathbb{E}_{\epsilon_1, \dots, \epsilon_k} \left[\log \frac{1}{k} \sum_{i=1}^k w(\mathbf{x}, \mathbf{h}(\mathbf{x}, \epsilon_i, \theta), \theta) \right] \\ &= \mathbb{E}_{\epsilon_1, \dots, \epsilon_k} \left[\nabla_{\theta} \log \frac{1}{k} \sum_{i=1}^k w(\mathbf{x}, \mathbf{h}(\mathbf{x}, \epsilon_i, \theta), \theta) \right] \end{aligned}$$

$$\begin{aligned} \widetilde{w}_i &= w_i / \sum_{i=1}^k w_i \\ &= \mathbb{E}_{\epsilon_1, \dots, \epsilon_k} \left[\sum_{i=1}^k \widetilde{w}_i \nabla_{\theta} \log w(\mathbf{x}, \mathbf{h}(\mathbf{x}, \epsilon_i, \theta), \theta) \right], \end{aligned}$$

$$\log p(\mathbf{x}) \geq \mathcal{L}_{k+1} \geq \mathcal{L}_k. \quad \sum_{i=1}^k \widetilde{w}_i \nabla_{\theta} \log w(\mathbf{x}, \mathbf{h}(\mathbf{x}, \epsilon_i, \theta), \theta).$$

Allows us to do approximate and quick Variational inference and make the bound tighter using the expensive Monte Carlo sampling (tradeoff of compute vs accuracy).

Application: Semi-supervised learning part1 [4]



Makes use of both labelled and unlabelled data

Application: Semi-supervised learning part1

Generative model

$$p(\mathbf{x}|y, \mathbf{z}) = f(\mathbf{x}; y, \mathbf{z}, \theta)$$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, I)$$

$$p(y|\pi) = \text{Cat}(y|\pi)$$

$$p(\pi) = \text{SymDir}(\alpha)$$

Recognition model

$$q_{\phi}(y, \mathbf{z}|\mathbf{x}) = q_{\phi}(\mathbf{z}|\mathbf{x})q_{\phi}(y|\mathbf{x})$$

$$q_{\phi}(y|\mathbf{x}) = \text{Cat}(y|\pi_{\phi}(\mathbf{x}))$$

$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu_{\phi}(\mathbf{x}), \text{diag}(\sigma_{\phi}^2(\mathbf{x})))$$

Loss function

$$\sum_{x \in D_{\text{unlab}}} [\mathbb{E}_{q_{\phi}(y|x)} [\mathcal{L}(x, y) - \log q_{\phi}(y|x)]] + \sum_{x, y \in D_{\text{lab}}} [\mathcal{L}(x, y) - \alpha \log q_{\phi}(y|x)]$$







P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel.
The helmholtz machine, 1995.



C. Doersch.
Tutorial on variational autoencoders.
arXiv preprint arXiv:1606.05908, 2016.



I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick,
S. Mohamed, and A. Lerchner.
**beta-vae: Learning basic visual concepts with a constrained
variational framework.**
2016.

-  D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling.
Semi-supervised learning with deep generative models.
In Advances in Neural Information Processing Systems, pages 3581–3589, 2014.
-  D. P. Kingma and M. Welling.
Auto-encoding variational bayes.
CoRR, abs/1312.6114, 2013.
-  K. G. Kirby.
A Tutorial on Helmholtz Machines.
2006.
-  T. A. Le.
Reinforce trick.
<http://www.tuananhle.co.uk/notes/reinforce.html>.



A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey.

Adversarial autoencoders.

arXiv preprint arXiv:1511.05644, 2015.



D. J. Rezende, S. Mohamed, and D. Wierstra.

Stochastic backpropagation and approximate inference in deep generative models.

In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Beijing, China, 22–24 Jun 2014. PMLR.