Deep Generative Models

Chapter 5: Variational Inference and VAEs

Ali Bereyhi

ali.bereyhi@utoronto.ca

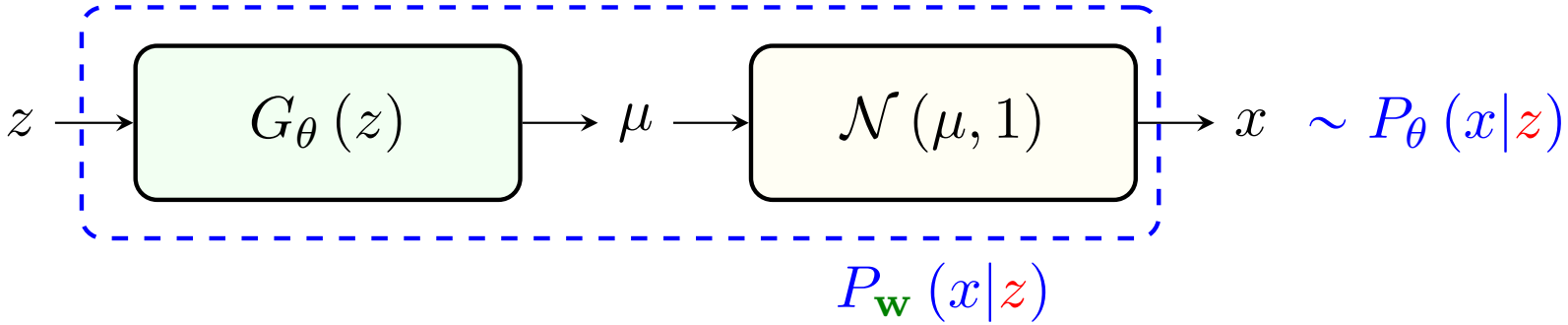
Department of Electrical and Computer Engineering

University of Toronto

Summer 2025

Recap: *Probabilistic Generator*

Let's get back to our original problem: *we have a probabilistic generator*



Mean μ is computed from latent $z \rightsquigarrow x$ is sampled from $\mathcal{N}(\mu, 1)$

Latent-space Generator

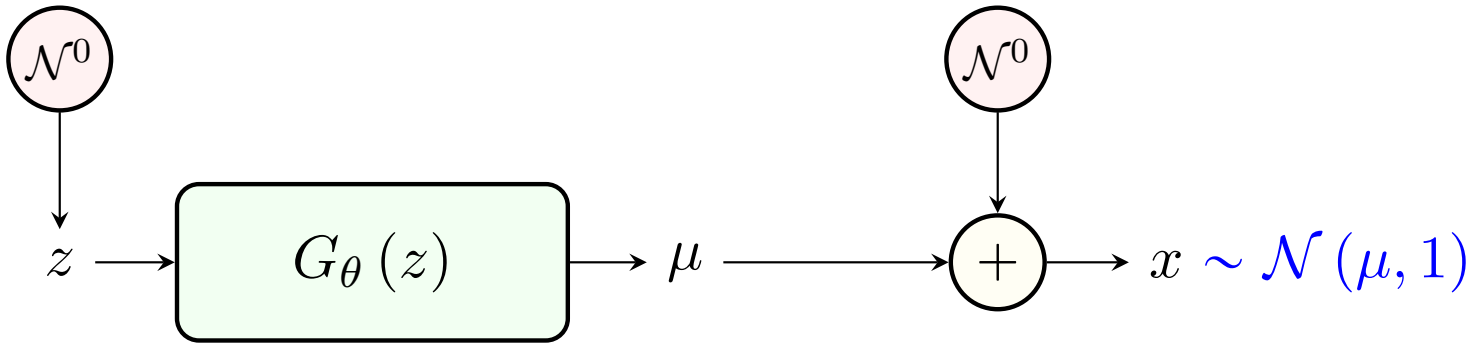
The sample x is generated conditionally from latent z as

$$P_\theta(x|z) = \mathcal{N}(G_\theta(z), 1)$$

Recap: *Sampling Probabilistic Generator*

Sampling is exactly as in *flow-based models*

1. *Sample latent as* $z \sim \mathcal{N}(0, 1) \equiv \mathcal{N}^0$
   - ↳ *Simply generate* m *independent Gaussian scalars*
2. *Pass forward latent through the model as* $\mu = G_\theta(z)$
3. *Sample data as* $x \sim \mathcal{N}(\mu, 1)$
   - ↳ *Simply sample* $\varepsilon \sim \mathcal{N}^0$
   - ↳ *Compute* $x = \mu + \varepsilon$



MLE Training: *Formulation*

Let's consider training via MLE: *say we have a batch of data samples*

$$\mathbb{D} = \left\{x^j \sim P_{\text{data}} : j = 1, \ldots, n\right\}$$

In MLE we maximize log-likelihood which is

$$\begin{aligned}
\max_{\theta} \log \mathcal{L}(\theta) &= \max_{\theta} \frac{1}{n} \sum_{j} \log P_\theta\left(x^j\right) \\
&= \max_{\theta} \hat{\mathbb{E}}_{x \sim P_{\text{data}}} \left\{\log P_\theta(x)\right\} \\
&= \max_{\theta} \hat{\mathbb{E}}_{x \sim P_{\text{data}}} \left\{\log \int P_\theta(x|z)\, P(z)\, \mathrm{d}z\right\}
\end{aligned}$$

to handle this challenging computation, we developed ELBO

MLE Training via *Variational Inference*

Computational ELBO

Variational inference indicates that

$$\log P_\theta\left(x^j\right) = \max_{\mathbf{w}} \mathrm{ELBO}_{\theta, x^j}\left(Q_{\mathbf{w}}\right)$$

for some computational model $Q_{\mathbf{w}}\left(z|x^j\right)$

- \+ *Why do we index ELBO by θ and x^j?*
- – Well! Just write the ELBO down

Recall that ELBO is given by

$$\mathrm{ELBO}_{\theta, x^j}\left(Q_{\mathbf{w}}\right) = \mathbb{E}_{z\sim Q_{\mathbf{w}}}\left\{\log P_\theta\left(x^j|z\right)\right\} - D_{\mathrm{KL}}\left(Q_{\mathbf{w}}\|\mathcal{N}^0\right)$$

which obviously depends on the generator P_θ and data sample x^j

MLE Training via *Variational Inference*

Let us now represent sample *ELBO estimate computed at* x *as*

$$\mathrm{ELBO}\left(\mathbf{w}, \theta | x\right) = \hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{\log P_{\theta}\left(x|z\right)\right\} - D_{\mathrm{KL}}\left(Q_{\mathbf{w}} \| \mathcal{N}^{0}\right)$$

We maximize this sample ELBO over $\mathbf{w}$ *to find log-likelihood; thus, MLE is*

$$\begin{aligned}\max_{\theta} \log \mathcal{L}\left(\theta\right) &= \max_{\theta} \hat{\mathbb{E}}_{x \sim P_{\mathrm{data}}} \left\{\log P_{\theta}\left(x\right)\right\} \\ &= \max_{\theta} \hat{\mathbb{E}}_{x \sim P_{\mathrm{data}}} \left\{\max_{\mathbf{w}} \mathrm{ELBO}\left(\mathbf{w}, \theta | x\right)\right\}\end{aligned}$$

Let's think algorithmic about computing log-likelihood

Full Loop via *Variational Inference*

$$\log \mathcal{L}(\theta) = \hat{\mathbb{E}}_{x \sim P_{\text{data}}} \left\{ \max_{\mathbf{w}} \text{ELBO}(\mathbf{w}, \theta | x) \right\}$$

`Compute_LL(`$\mathbf{w}, \theta$`):`

1: *Sample a batch of data samples* $\{x^j : j = 1, \ldots, n\}$
2: **for** $j = 1, \ldots, n$ **do**
3: *Sample a batch of latent samples* $\{z^\ell : \ell = 1, \ldots, r\}$ *from* $Q_{\mathbf{w}}$
4: *Compute* $\text{ELBO}(\mathbf{w}, \theta | x^j)$ *over latent batch*
5: **for** T *iterations* **do**
6: *Iterate with* $\nabla_{\mathbf{w}} \text{ELBO}(\mathbf{w}, \theta | x^j)$ *to maximize ELBO at* $\mathbf{w}^\star$
7: **end for**
8: *Set* $\text{ELBO}^j \leftarrow \text{ELBO}(\mathbf{w}^\star, \theta | x^j)$
9: **end for**
10: *Estimate log-likelihood as* $\log \mathcal{L} \leftarrow$ `mean` $\{\text{ELBO}^j\}$
11: **return** *log-likelihood estimate* $\log \mathcal{L}$

This means that log-likelihood optimization requires a two-tier loop

↳ *This is computationally expensive*

Approximate Loop via *Variational Inference*

Using Jensen's inequality: *we can write*

$$\hat{\mathbb{E}}_{x\sim P_{\text{data}}}\left\{\max_{\mathbf{w}} \text{ELBO}\left(\mathbf{w},\theta|x\right)\right\} \geqslant \max_{\mathbf{w}} \hat{\mathbb{E}}_{x\sim P_{\text{data}}}\left\{\text{ELBO}\left(\mathbf{w},\theta|x\right)\right\}$$

This means that the MLE loop can be bounded as

$$\begin{aligned}\max_{\theta} \log \mathcal{L}\left(\theta\right) &= \max_{\theta} \hat{\mathbb{E}}_{x\sim P_{\text{data}}}\left\{\max_{\mathbf{w}} \text{ELBO}\left(\mathbf{w},\theta|x\right)\right\} \\ &\geqslant \max_{\theta}\max_{\mathbf{w}} \hat{\mathbb{E}}_{x\sim P_{\text{data}}}\left\{\text{ELBO}\left(\mathbf{w},\theta|x\right)\right\}\end{aligned}$$

Hoping that this bound is tight enough, we could simplify the MLE loop as

$$\max_{\theta} \log \mathcal{L}\left(\theta\right) \approx \max_{\theta}\max_{\mathbf{w}} \hat{\mathbb{E}}_{x\sim P_{\text{data}}}\left\{\text{ELBO}\left(\mathbf{w},\theta|x\right)\right\}$$

Approximate Loop via *Variational Inference*

`Approx_Compute_LL(`$\mathbf{w}, \theta$`):`

1: *Sample a batch of data samples* $\left\{x^j : j = 1, \ldots, n\right\}$
2: **for** $j = 1, \ldots, n$ **do**
3: *Sample a batch of latent samples* $\left\{z^\ell : \ell = 1, \ldots, r\right\}$ *from* $Q_{\mathbf{w}}$
4: *Compute* $\mathrm{ELBO}\left(\mathbf{w}, \theta | x^j\right)$ *over latent batch*
5: *Set* $\mathrm{ELBO}^j \leftarrow \mathrm{ELBO}\left(\mathbf{w}, \theta | x^j\right)$ `#no inner optimization loop`
6: **end for**
7: *Estimate log-likelihood as* $\log \mathcal{L} \leftarrow$ `mean` $\left\{\mathrm{ELBO}^j\right\}$
8: **return** *log-likelihood estimate* $\log \mathcal{L}$

This makes log-likelihood optimization a classical risk minimization with

$$\hat{R}\left(\mathbf{w}, \theta\right) = \hat{\mathbb{E}}_{x \sim P_{\text{data}}} \left\{-\mathrm{ELBO}\left(\mathbf{w}, \theta | x\right)\right\}$$

which can be readily performed by a single gradient iteration loop

Completing Computational Model

Though the training is formulated: *there are a few questions*

+ *How should we build the model* $Q_{\mathbf{w}}(z|x)$*?*
– *It needs to be a probability distribution computed from a sample* x
– *It should marginalize to prior latent distribution*
+ *How easy is it to compute the required sample gradients, i.e.,*

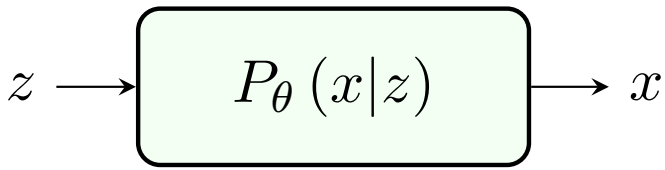
$$\nabla_{\mathbf{w}}\text{ELBO}(\mathbf{w}, \theta|x) \qquad \textit{and} \qquad \nabla_{\theta}\text{ELBO}(\mathbf{w}, \theta|x)$$

– *Well! They are mostly trivial except a tiny tricky part*

Probabilistic Generation: *Decoding*

The computational model given by this *probabilistic generation* describes an

$$\textit{Autoencoder (AE)} \equiv \textit{encoder } x \mapsto z \textit{ + decoder } z \mapsto x$$



Decoder

The decoder is indeed probabilistic generator $P_\theta(x|z)$ *which consists of*

1. *a computational model* $G_\theta : \mathbb{R}^m \mapsto \mathbb{R}^d$
2. *a Gaussian sampler that decodes latent* z *by sampling*

$$P_\theta(x|z) = C \exp\left\{-\frac{\|x - G_\theta(z)\|^2}{2}\right\}$$

Learning Latent Distribution: *Encoding*

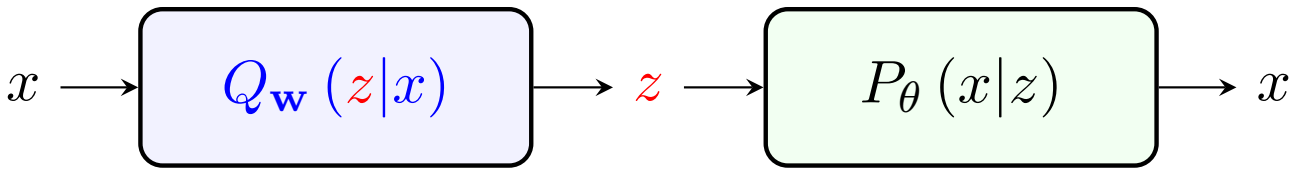
To computationally model $Q_{\mathbf{w}}(z|x)$: *we note that this model*

- *takes x as input and returns a $Q_{\mathbf{w}}(z|x)$ that we can sample*
- *approximates $P(z|x)$ meaning that it should marginalize to latent prior*

$$\int Q_{\mathbf{w}}(z|x) P_{\text{data}}(x)\,\mathrm{d}x \approx \int P(z|x)\, P_{\text{data}}(x)\,\mathrm{d}x = P(z)$$

The model should be capable of giving such approximation!

This model describes an encoder which encodes x to its latent representation



Modeling Encoder

+ *How can we design a model that marginalizes to $\mathcal{N}^0$?!*

Recall: *Universality of Gaussian Mixtures*

Gaussian mixture model describes a dense set of distributions

Let's take another look: *we want to set $Q_{\mathbf{w}}(z|x)$ such that*

$$\hat{P}_{\mathbf{w}}(z) = \int Q_{\mathbf{w}}(z|x) P_{\text{data}}(x)\, \mathrm{d}x$$

be a good approximator of latent prior $\mathcal{N}^0$ $\rightsquigarrow$ setting

$$Q_{\mathbf{w}}(z|x) = \mathcal{N}(\eta_{\mathbf{w}}(x), \Sigma_{\mathbf{w}}(x))$$

we get a Gaussian mixture $\hat{P}_{\mathbf{w}}(z)$

↳ *It approximates most distributions including $\mathcal{N}^0$*

Encoding via *Gaussian Model*

+ *How can we design a model that marginalizes to $\mathcal{N}^0$?!*
– We just set it to be a Gaussian whose mean and covariance are learned

Encoder

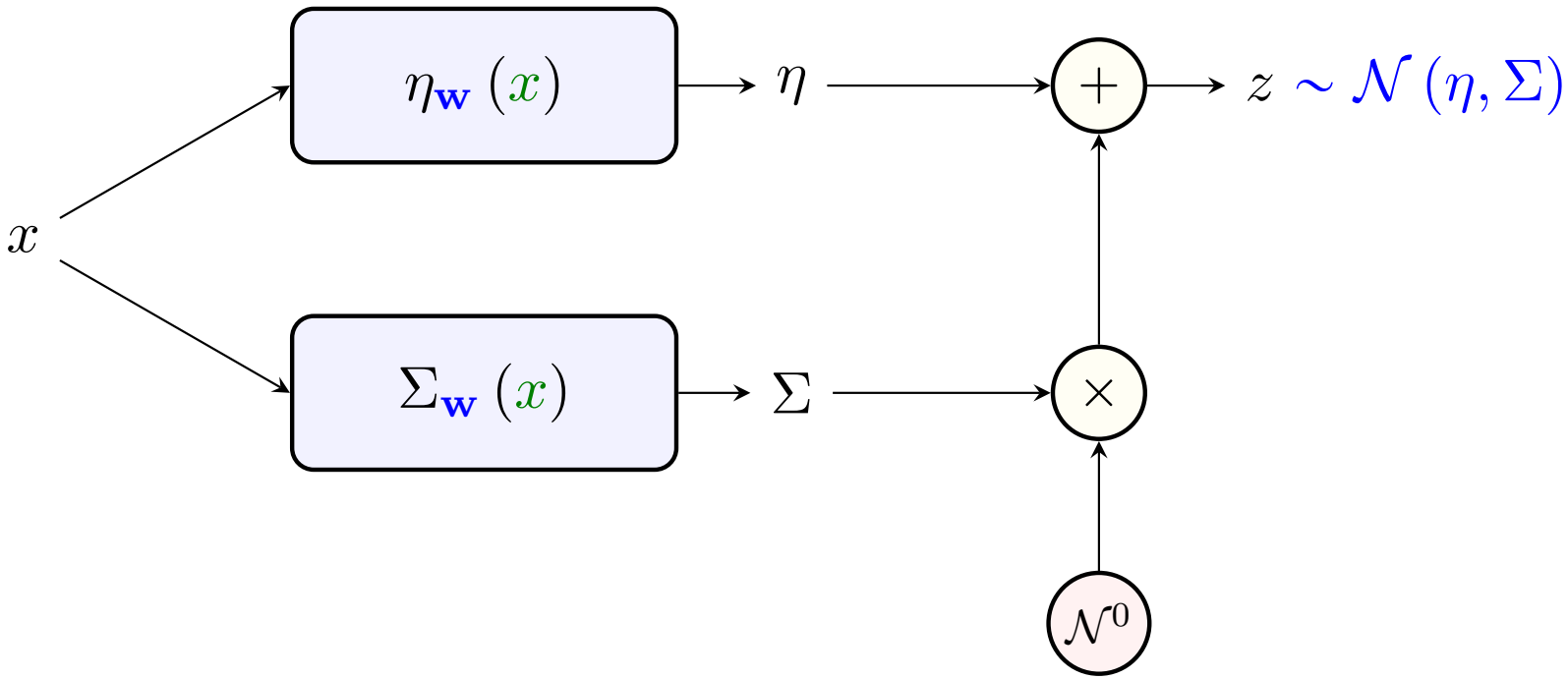
The encoder $Q_{\mathbf{w}}(z|x)$ consists of

1. *a computational model $\eta_{\mathbf{w}} : \mathbb{R}^d \mapsto \mathbb{R}^m$ that computes the mean*
2. *a computational model $\Sigma_{\mathbf{w}} : \mathbb{R}^d \mapsto \mathbb{R}^{m \times m}$ that computes covariance*
   - ! *Covariance should be positive semi-definite*
   - ↳ *The model should always compute a positive semi-definite output*
3. *a Gaussian sampler that encodes data x by sampling*

$$Q_{\mathbf{w}}(z|x) = \mathcal{N}(\eta_{\mathbf{w}}(x), \Sigma_{\mathbf{w}}(x))$$

Encoding via *Gaussian Model*

Such a model is easy to realize



Encoding: *Building Positive-Definite Covariance*

In practice: *we usually learn a diagonal covariance*

$$\Sigma = \operatorname{diag}\left\{\sigma_1^2, \ldots, \sigma_m^2\right\}$$

This corresponds to z with independent entries whose distribution is

$$Q_{\mathbf{w}}(z|x) = \frac{1}{\prod_i \sqrt{2\pi\sigma_{\mathbf{w},i}^2(x)}} \exp\left\{-\sum_i \frac{(z_i - \eta_{\mathbf{w},i}(x))^2}{2\sigma_{\mathbf{w},i}^2(x)}\right\}$$

This is preferred computationally because

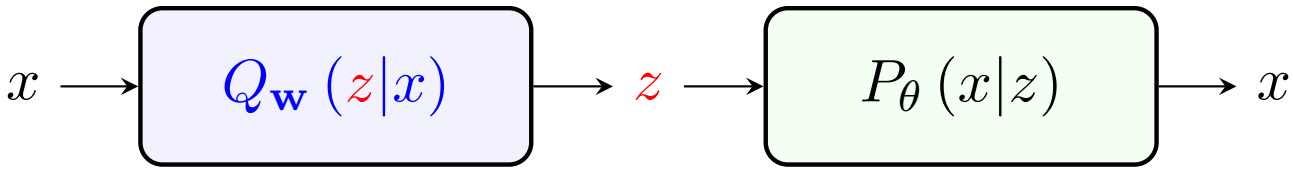
- *It is readily realized using $\Sigma_{\mathbf{w}} : \mathbb{R}^d \mapsto \mathbb{R}_+^m$ and setting $\Sigma = \operatorname{diag}\{\Sigma_{\mathbf{w}}(x)\}$*
- *It guarantees that the covariance is positive-definite*
- *It is much easier to sample from*
  - ↳ *since the entries of z are independent*

Variational AE: *End-to-End Architecture*

General VAE

A variational autoencoder (VAE) consists of

- *an encoder which learns latent by sampling a conditional Gaussian*
- *a decoder that generates from latent by sampling a Gaussian generator*



! Attention

Similar to discriminator of GANs, the encoder in VAEs

↳ *is used to implicitly compute the log-likelihood*

↳ *is used only for training and is not required for generation*

However, a trained encoder can also give us latent representation if we want

VAE: *Few Notes*

+ *Should we always consider Gaussian encoder and decoder?*
- *It's not a must! We can use other encoder and decoders, but*
  - ↳ *universality tells us that Gaussian is good enough*
  - ↳ *Gaussian distribution is easy to sample*
+ *Why don't we learn the covariance in decoder?*
- *In theory we can do that! We don't do it in practice though since*
  - ↳ *we might face numerical challenges while training VAEs*
  - ↳ *We will get some idea about this in Assignment 3*

Training VAEs: *ELBO Estimate*

For training, we maximize ELBO: *the ELBO estimate is given by*

$$\mathrm{ELBO}\left(\mathbf{w}, \theta | x\right) = \hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{\log P_{\theta}\left(x|z\right)\right\} - D_{\mathrm{KL}}\left(Q_{\mathbf{w}} \| \mathcal{N}^{0}\right)$$

Let's look at the first term: considering Gaussian generator, we have

$$\begin{aligned}\log P_{\theta}\left(x|z\right) &= \log C + \log \exp\left\{-\frac{\|x - G_{\theta}\left(z\right)\|^{2}}{2}\right\} \\ &= C' + -\frac{\|x - G_{\theta}\left(z\right)\|^{2}}{2}\end{aligned}$$

Therefore, the first term is given by

$$\hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{\log P_{\theta}\left(x|z\right)\right\} = C_{0} - \frac{1}{2}\hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{\|x - G_{\theta}\left(z\right)\|^{2}\right\}$$

Estimating ELBO in VAE

The second term is determined more compactly

$$
\begin{aligned}
D_{\mathrm{KL}}\left(Q_{\mathbf{w}} \| \mathcal{N}^0\right) &= \mathbb{E}_{z\sim Q_{\mathbf{w}}}\left\{\log \frac{Q_{\mathbf{w}}\left(z|x\right)}{\mathcal{N}^0\left(z\right)}\right\} \\
&= \mathbb{E}_{z\sim Q_{\mathbf{w}}}\left\{\log \frac{\exp\left\{-\sum_i \frac{\left(z_i-\eta_{\mathbf{w},i}\left(x\right)\right)^2}{2\sigma^2_{\mathbf{w},i}\left(x\right)}\right\}}{\prod_i \sqrt{2\pi\sigma^2_{\mathbf{w},i}\left(x\right)}} \frac{\left(\sqrt{2\pi}\right)^m}{\exp\left\{-\sum_i \frac{z_i^2}{2}\right\}}\right\} \\
&= \mathbb{E}_{z\sim Q_{\mathbf{w}}}\left\{-\frac{1}{2}\sum_i\left[\log\sigma^2_{\mathbf{w},i}\left(x\right)+\frac{\left(z_i-\eta_{\mathbf{w},i}\left(x\right)\right)^2}{\sigma^2_{\mathbf{w},i}\left(x\right)}-z_i^2\right]\right\} \\
&= -\frac{1}{2}\sum_i \log\sigma^2_{\mathbf{w},i}\left(x\right)+1-\eta^2_{\mathbf{w},i}\left(x\right)-\sigma^2_{\mathbf{w},i}\left(x\right)
\end{aligned}
$$

Estimating ELBO in VAE

Let us define this simple function: $A(\cdot) : \mathbb{R}^m \mapsto \mathbb{R}$ *is*

$$A(u) = \sum_i \log \frac{1}{u_i} + u_i$$

We can then say

The second term is given by

$$D_{\mathrm{KL}}\left(Q_{\mathbf{w}} \| \mathcal{N}^0\right) = \frac{1}{2} A\left(\Sigma_{\mathbf{w}}(x)\right) + \frac{1}{2}\|\eta_{\mathbf{w}}(x)\|^2 - \frac{m}{2}$$

This term only depends on the mean and covariance networks!

Training VAEs: *Risk Function*

MLE in VAEs *is equivalent to ELBO maximization*

$$\max_{\theta} \log \mathcal{L}(\theta) \approx \max_{\theta} \max_{\mathbf{w}} \hat{\mathbb{E}}_{x \sim P_{\text{data}}} \{\text{ELBO}(\mathbf{w}, \theta | x)\}$$

If we drop constants and coefficients, the sample risk reduces to

$$\begin{aligned} R(\mathbf{w}, \theta | x) &\propto -\text{ELBO}(\mathbf{w}, \theta | x) \\ &= \hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{ \|x - G_{\theta}(z)\|^2 \right\} + A(\Sigma_{\mathbf{w}}(x)) + \|\eta_{\mathbf{w}}(x)\|^2 \end{aligned}$$

MLE via ELBO Maximization

MLE training is equivalent to the following empirical risk minimization

$$\min_{\mathbf{w}, \theta} \hat{R}(\mathbf{w}, \theta) = \min_{\mathbf{w}, \theta} \hat{\mathbb{E}}_{x \sim P_{\text{data}}} \{R(\mathbf{w}, \theta | x)\}$$

ELBO Maximization by *Gradient Descent*

Numerically, *we are going to use a gradient-based optimizer which needs*

$$\nabla_{\mathbf{w}} R\left(\mathbf{w}, \theta | x\right) \qquad \textit{and} \qquad \nabla_{\theta} R\left(\mathbf{w}, \theta | x\right)$$

Let's start looking into $\nabla_\theta R$*: we can expand the gradient as*

$$\begin{aligned}
\nabla_\theta R &= \nabla_\theta \hat{\mathbb{E}}_{z\sim Q_{\mathbf{w}}} \left\{ \|x - G_\theta\left(z\right)\|^2 \right\} + \underbrace{\nabla_\theta A\left(\Sigma_{\mathbf{w}}\left(x\right)\right)}_{0} + \underbrace{\nabla_\theta \|\eta_{\mathbf{w}}\left(x\right)\|^2}_{0} \\
&= \hat{\mathbb{E}}_{z\sim Q_{\mathbf{w}}} \left\{ \nabla_\theta \|x - G_\theta\left(z\right)\|^2 \right\} \\
&= \hat{\mathbb{E}}_{z\sim Q_{\mathbf{w}}} \left\{ \nabla_\mu \|x - \mu\|^2 \Big|_{\mu = G_\theta(z)} \circ \underbrace{\nabla_\theta G_\theta\left(z\right)}_{\text{Backpropagation}} \right\}
\end{aligned}$$

This is readily computed by backpropagation over the model G_θ ✔

ELBO Maximization by *Gradient Descent*

It's good to think about it algorithmic

`Compute_Grad_Dec`$(x, Q_{\mathbf{w}}, G_\theta)$

1: *Sample a batch of latent samples* $\left\{z^\ell : \ell = 1, \ldots, n\right\}$ *from* $Q_{\mathbf{w}}\left(z|x\right)$
2: **for** $\ell = 1, \ldots, n$ **do**
3: *Pass* z^ℓ *forward and backward over* G_θ
4: *Use chain rule to compute* $\nabla^\ell = \nabla_\theta \|x - G_\theta\left(z^\ell\right)\|^2$
5: **end for**
6: *Estimate gradient w.r.t. decoder as* $\nabla_\theta R\left(\mathbf{w}, \theta|x\right) = \texttt{mean}\left\{\nabla^\ell\right\}$
7: **return** *Gradient* $\nabla_\theta R$ *at sample* x

ELBO Maximization by *Gradient Descent*

Now, let's look into $\nabla_{\mathbf{w}} R$: *we can expand the gradient as*

$$\nabla_{\mathbf{w}} R = \underbrace{\nabla_{\mathbf{w}} \hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{ \|x - G_\theta(z)\|^2 \right\}}_{?} + \underbrace{\nabla_{\mathbf{w}} A\left(\Sigma_{\mathbf{w}}(x)\right)}_{\text{Backprop}} + \underbrace{\nabla_{\mathbf{w}} \|\eta_{\mathbf{w}}(x)\|^2}_{\text{Backprop}}$$

- ✔ *Latter two are computed by basic backpropagation*
- ✖ *Former is though not a conventional gradient computation*

ELBO Maximization by *Gradient Descent*

Note that in the former, the gradient cannot go inside the expectation

$$\begin{aligned}\nabla_{\mathbf{w}}\mathbb{E}_{z\sim Q_{\mathbf{w}}}\left\{\|x-G_\theta(z)\|^2\right\} &= \nabla_{\mathbf{w}}\int\|x-G_\theta(z)\|^2 Q_{\mathbf{w}}(z|x)\,\mathrm{d}z\\ &= \int\|x-G_\theta(z)\|^2\nabla_{\mathbf{w}}Q_{\mathbf{w}}(z|x)\,\mathrm{d}z\\ &\neq \mathbb{E}_{z\sim Q_{\mathbf{w}}}\left\{\nabla_{\mathbf{w}}\|x-G_\theta(z)\|^2\right\}\end{aligned}$$

Naively speaking, we have

$$\mathbb{E}_{z\sim Q_{\mathbf{w}}}\left\{\nabla_{\mathbf{w}}\|x-G_\theta(z)\|^2\right\}=0$$

But indeed z is a stochastic function of $\mathbf{w}$ through its distribution!

+ *How can we compute the gradient of a stochastic function?!*

Gradient of Stochastic Function

We can formulate the problem as follows: *say we have*

$$R(\mathbf{w}) = \mathbb{E}_{z \sim Q_{\mathbf{w}}} \{F(z)\}$$

Function $R(\mathbf{w})$ *depends on* $\mathbf{w}$ *through samples* $z \sim Q_{\mathbf{w}}$*:its gradient reads*

$$\begin{aligned} \nabla_{\mathbf{w}} R(\mathbf{w}) &= \nabla_{\mathbf{w}} \mathbb{E}_{z \sim Q_{\mathbf{w}}} \{F(z)\} \\ &= \nabla_{\mathbf{w}} \int F(z) Q_{\mathbf{w}}(z) \, \mathrm{d}z \end{aligned}$$

which can be computed by two tricks; namely,

1. *Importance Sampling*
2. *Reparameterization Trick*

Importance Sampling

Let's open up the expression: *we can write*

$$\nabla_{\mathbf{w}} R(\mathbf{w}) = \int F(z) \nabla_{\mathbf{w}} Q_{\mathbf{w}}(z) \, \mathrm{d}z$$

We can use the simple fact that

$$\nabla_{\mathbf{w}} \log Q_{\mathbf{w}}(z) = \frac{\nabla_{\mathbf{w}} Q_{\mathbf{w}}(z)}{Q_{\mathbf{w}}(z)}$$

Using this fact, we can write the gradient as

$$\begin{aligned} \nabla_{\mathbf{w}} R(\mathbf{w}) &= \int F(z) \nabla_{\mathbf{w}} \log Q_{\mathbf{w}}(z) \, Q_{\mathbf{w}}(z) \mathrm{d}z \\ &= \mathbb{E}_{z \sim Q_{\mathbf{w}}} \{ F(z) \nabla_{\mathbf{w}} \log Q_{\mathbf{w}}(z) \} \end{aligned}$$

Importance Sampling

Computing Gradient by *Importance Sampling*

For any function F and distribution $Q_{\mathbf{w}}$, we can estimate gradient as

$$\nabla_{\mathbf{w}} \mathbb{E}_{z \sim Q_{\mathbf{w}}} \{F(z)\} = \hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \{F(z) \nabla_{\mathbf{w}} \log Q_{\mathbf{w}}(z)\}$$

`ImpSampling`$(Q_{\mathbf{w}})$

1: *Sample a batch of samples* $\{z^\ell : \ell = 1, \ldots, n\}$ *from* $Q_{\mathbf{w}}(z)$
2: **for** $\ell = 1, \ldots, n$ **do**
3: *Pass* z^ℓ *forward and backward over* $\log Q_{\mathbf{w}}$
4: **end for**
5: *Estimate the gradient as* $\nabla_{\mathbf{w}} R = \texttt{mean}\{F(z^\ell) \nabla_{\mathbf{w}} \log Q_{\mathbf{w}}(z^\ell)\}$
6: **return** *Gradient* $\nabla_{\mathbf{w}} R$

Reparameterization Trick

There is also an alternative trick: *say we could find some* $\varepsilon \sim Q_0$ *such that*

$$z = f_{\mathbf{w}}(\varepsilon)$$

is distributed by $z \sim Q_{\mathbf{w}}$*: we can then write*

$$\begin{aligned}
\nabla_{\mathbf{w}} R(\mathbf{w}) &= \nabla_{\mathbf{w}} \mathbb{E}_{z \sim Q_{\mathbf{w}}} \{F(z)\} \\
&= \nabla_{\mathbf{w}} \mathbb{E}_{\varepsilon \sim Q_0} \{F(f_{\mathbf{w}}(\varepsilon))\} \\
&= \mathbb{E}_{\varepsilon \sim Q_0} \left\{ \nabla_z F(z = f_{\mathbf{w}}(\varepsilon)) \circ \underbrace{\nabla_{\mathbf{w}} f_{\mathbf{w}}(\varepsilon)}_{\text{Backprop}} \right\}
\end{aligned}$$

which we can be readily estimated by sampling

Reparameterization Trick

Computing Gradient by *Reparameterization*

Assuming $z = f_{\mathbf{w}}(\varepsilon)$ *for some* $\varepsilon \sim Q_0$, *we can estimate gradient as*

$$\nabla_{\mathbf{w}} \mathbb{E}_{z \sim Q_{\mathbf{w}}} \{F(z)\} = \hat{\mathbb{E}}_{\varepsilon \sim Q_0} \{\nabla_z F(z = f_{\mathbf{w}}(\varepsilon)) \circ \nabla_{\mathbf{w}} f_{\mathbf{w}}(\varepsilon)\}$$

`ReparamTrick`$(Q_{\mathbf{w}} \leftrightsquigarrow f_{\mathbf{w}}, Q_0)$

1: *Sample a batch of samples* $\{\varepsilon^\ell : \ell = 1, \ldots, n\}$ *from* Q_0
2: **for** $\ell = 1, \ldots, n$ **do**
3: *Pass* ε^ℓ *forward and backward over* $f_{\mathbf{w}}$
4: **end for**
5: *Estimate the gradient as* $\nabla_{\mathbf{w}} R = \texttt{mean}\{\nabla_z F(f_{\mathbf{w}}(\varepsilon^\ell)) \circ \nabla_{\mathbf{w}} f_{\mathbf{w}}(\varepsilon^\ell)\}$
6: **return** *Gradient* $\nabla_{\mathbf{w}} R$

Training VAE: *Reparameterization Trick*

Back to our problem: *we need to compute*

$$\nabla_{\mathbf{w}} \mathbb{E}_{z \sim Q_{\mathbf{w}}} \left\{ \|x - G_\theta(z)\|^2 \right\}$$

We note that for a given x*, we can build* $z \sim Q_{\mathbf{w}}(z|x)$ *from* $\varepsilon \sim \mathcal{N}^0$ *as*

$$z = \eta_{\mathbf{w}}(x) + \sqrt{\Sigma_{\mathbf{w}}(x)}\, \varepsilon$$

So we can use reparameterization trick to estimate this gradient with

$$\hat{\mathbb{E}}_{\varepsilon \sim \mathcal{N}^0} \left\{ \nabla_z \|x - G_\theta(z)\|^2 \circ \left[\nabla_{\mathbf{w}} \eta_{\mathbf{w}}(x) + \varepsilon \circ \nabla_{\mathbf{w}} \sqrt{\Sigma_{\mathbf{w}}(x)} \right] \right\}$$

which is computed by sampling ✔

Training with ELBO Maximization

Recall that *we needed*

$$\nabla_{\mathbf{w}} R\left(\mathbf{w}, \theta | x\right) \qquad \textit{and} \qquad \nabla_{\theta} R\left(\mathbf{w}, \theta | x\right)$$

which are now given by

- *With respect to θ, we use standard backpropagation*

$$\nabla_{\theta} R = \hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{ \nabla_{\mu} \| x - G_{\theta}\left(z\right) \|^2 \circ \nabla_{\theta} G_{\theta}\left(z\right) \right\}$$

- *With respect to $\mathbf{w}$, we use backpropagation and reparameterization*

$$\nabla_{\mathbf{w}} R = \underbrace{\nabla_{\mathbf{w}} \hat{\mathbb{E}}_{z \sim Q_{\mathbf{w}}} \left\{ \| x - G_{\theta}\left(z\right) \|^2 \right\}}_{\text{reparameterization}} + \underbrace{\nabla_{\mathbf{w}} A\left(\Sigma_{\mathbf{w}}\left(x\right)\right)}_{\text{Backprop}} + \underbrace{\nabla_{\mathbf{w}} \| \eta_{\mathbf{w}}\left(x\right) \|^2}_{\text{Backprop}}$$

VAE: *Training Loop*

`Train_VAE(`$Q_{\mathbf{w}}$`:Enc,`P_θ`:Dec)`

Initiate the decoder $P_\theta \equiv G_\theta$ *and encoder* $Q_{\mathbf{w}} \equiv (\eta_{\mathbf{w}}, \Sigma_{\mathbf{w}})$ *with some* θ and $\mathbf{w}$
for *multiple epochs* **do**
  Sample a batch of data samples $\{x^j : j = 1, \ldots, n\}$ *from* $\mathbb{D}$
  for $j = 1, \ldots, n$ **do**
    Pass x^j *forward the encoder to compute* η^j *and* Σ^j
    Sample $\{\varepsilon^\ell : \ell = 1, \ldots, k\}$ *from* $\mathcal{N}^0$ *and compute latent* $z^\ell = \eta^j + \sqrt{\Sigma^j}\varepsilon^\ell$
    Compute $\nabla_{\mathbf{w}} R^j$ *by backpropagation and reparameterization with* ε^ℓ
    for $\ell = 1, \ldots, k$ **do**
      Pass latent z^ℓ *forward and backward the decoder*
    end for
    Compute $\nabla_\theta R^j$ *averaging backpropagation on decoder*
  end for
  Update $\mathbf{w}$ *using* `Opt_avg` $\{\nabla_{\mathbf{w}} R^j\}$
  Update θ *using* `Opt_avg` $\{\nabla_\theta R^j\}$
end for
return *trained encoder* $(\eta_{\mathbf{w}}, \Sigma_{\mathbf{w}})$ *and decoder* G_θ

VAE: *Generation*

`Sample_VAE(`$G_{\mathbf{w}}$`:Dec)`

1: *Sample latent* $z \sim \mathcal{N}^0$
2: *Compute mean value* $\mu = G_{\mathbf{w}}(z)$
3: *Sample* $\varepsilon \sim \mathcal{N}^0$
4: *Compute generated sample as* $x = \mu + \varepsilon$
5: **return** *Sample* x

Comparing VAEs to Vanilla AEs

+ *Why we call them autoencoder?! It looks different than what we do with vanilla AEs!*
– Indeed, vanilla AEs are the special case of them

Consider the case where *we set the encoder to be deterministic*

$$Q_{\mathbf{w}}(z|x) = \begin{cases} 1 & z = \eta_{\mathbf{w}}(x) \\ 0 & z \neq \eta_{\mathbf{w}}(x) \end{cases}$$

if we forget about the KL divergence in the loss, we have

$$\begin{aligned} R(\mathbf{w}, \theta|x) &= \hat{\mathbb{E}}_{z\sim Q_{\mathbf{w}}} \left\{ \|x - G_\theta(z)\|^2 \right\} \\ &= \|x - G_\theta(\eta_{\mathbf{w}}(x))\|^2 \end{aligned}$$

Comparing VAEs to Vanilla AEs

This sample loss

$$R\left(\mathbf{w}, \theta | x\right) = \left\| x - G_\theta\left(\eta_{\mathbf{w}}\left(x\right)\right)\right\|^2$$

describes a vanilla AE with deterministic encoder and decoder

Intuitive Connection to AEs

VAE is an AE with probabilistic encoder and decoder

VAEs: *Wrap Up*

In a nutshell, VAEs

- *use a probabilistic generator to sample data distribution*
- *invoke variational inference to implicitly maximize likelihood*
  - ↳ *an encoder learns the posterior latent distribution*
  - ↳ *MLE is performed by maximizing ELBO*
  - ↳ *reparameterization trick is invoked to estimate gradient*

As compared to GANs

- ✔ *VAEs are more stable while training*
- ✔ *VAEs are able to compute latent representation of data*
  - ↳ *Trained encoder can encode data samples in latent space*
- ✖ *Samples of VAE are less sharp as compared to GAN*
  - ↳ *This is due to probabilistic generation in VAEs*

Next stop: some well-known VAEs