Notes on Variational Autoencoders

David Meyer

dmm@{1-4-5.net,uoregon.edu,...}

17 Jan 2016

1 Introduction

Variational Autoecoders (VAEs) [1] are generative models in which we have examples X that are distributed according to some unknown distribution $P_{gen}(X)$, and our goal is to learn a model P which we can sample from, such that P is as similar as possible to P_{gen}. This is a long standing problem in machine learning, where most approaches have had three significant drawbacks. First, they might require strong assumptions about the structure in the data. Second, they might make severe approximations, leading to suboptimal models. Third, they might rely on computationally expensive inference procedures like Markov Chain Monte Carlo (MCMC). Recent progress in training neural networks as powerful function approximators through backpropagation have provided new ways to think about the problem. VAEs are one such method.

2 Review: Latent Variable Models

Most of what we'll review in this section is based on two simple but fundamental rules of probability:

$$\text{Sum Rule:} \qquad p(\mathcal{X}) = \sum_y p(\mathcal{X} \cap \mathcal{Y}) \tag{1}$$

$$\text{Product Rule:} \qquad p(\mathcal{X}, \mathcal{Y}) = p(\mathcal{X}|\mathcal{Y})p(\mathcal{Y}) \tag{2}$$

Note that

$$p(\Theta, \mathcal{X}) = p(\mathcal{X}, \Theta) \tag{3}$$

$$p(\Theta, \mathcal{X}) = p(\Theta|\mathcal{X})p(\mathcal{X}) \tag{4}$$

$$p(\mathcal{X}, \Theta) = p(\mathcal{X}|\Theta)p(\Theta) \tag{5}$$

$$p(\Theta|\mathcal{X})p(\mathcal{X}) = p(\mathcal{X}|\Theta)p(\Theta) \tag{6}$$

Solving for $p(\Theta|\mathcal{X})$ we get Bayes' Theorem

$$p(\Theta|\mathcal{X}) = \frac{p(\mathcal{X}|\Theta)p(\Theta)}{p(\mathcal{X})} \tag{7}$$

You might also see Bayes' Theorem written using the *Law of Total Probability*[1] which is sometimes written as follows:

$$\begin{aligned} p(A) &= \sum_n p(A \cap B_n) && \text{\# by the } \textit{Sum Rule} \text{ (Equation 1)} && (8)\\ &= \sum_n p(A, B_n) && \text{\# in the notation used in Equation 1} && (9)\\ &= \sum_n p(A|B_n)p(B_n) && \text{\# by the } \textit{Product Rule} \text{ (Equation 2)} && (10) \end{aligned}$$

2.1 Latent Variables?

The basic idea here is that the distribution of the data we observe, $\mathcal{X}$, is controlled (or at least significantly influenced) by a set of hidden or *latent* variables $\mathcal{Z}$. In many cases what we want to learn is a mapping from the latent variables z to some complicated distribution on x.

Stated more formally, for $x \in \mathcal{X}$ and $z \in \mathcal{Z}$ (z is a vector of latent variables, generally with values taken from $\mathbb{R}$):

$$\begin{aligned} p(\mathbf{x}) &= \int_z p(\mathbf{x}, \mathbf{z}) dz && && (11)\\ &= \int_z p(\mathbf{x}|\mathbf{z})p(\mathbf{z})dz && \text{\# sum and product rules} && (12) \end{aligned}$$

where $p(\mathbf{z})$ is something simple like $\mathcal{N}(0, I)$ and $p(\mathbf{x}|\mathbf{z}) = g(\mathbf{z})$ (we'll use $g(\mathbf{z})$ below).

$p(x)$ also sometimes called the *evidence* or the *marginal likelihood* since Bayes Rule/Theorem (Equation 7) can be characterized as

$$\text{posterior} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

However, computing the probability of the evidence is frequently intractable due to the integral shown in Equation 12. This is where we appeal to Approximate Inference.

[1] The Law of Total Probability is a combination of the Sum and Product Rules

3 The Problem of Approximate Inference

Let $\mathbf{x} = x_{1:N}$ be a set of observed variables and let $\mathbf{z} = z_{1:M}$ be a set of latent variables with joint distribution $p(\mathbf{z}, \mathbf{x})$. Then the inference problem is to compute the conditional distribution of latent variables given the observations, that is, $p(\mathbf{z}|\mathbf{x})$.

We can write the conditional or posterior distribution as

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} \tag{13}$$

The denominator of Equation 13 is the marginal distribution of the observations (also called the *evidence*), and is calculated by *marginalizing* out the latent variables from the joint distribution, i.e.,

$$p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}, \mathbf{x}) \, d\mathbf{z} \tag{14}$$

In many cases of interest this integral is not available in closed form or is intractable (requires exponential time to compute). However, the model *evidence* is just the quantity we need to compute the conditional ($p(\mathbf{z}|\mathbf{x})$) from the joint ($p(\mathbf{z}, \mathbf{x})$). This is why inference in these cases can be hard.

Recall that in variational inference we specify a family $\mathscr{L}$ of distributions over the latent variables. Each $q(\mathbf{z}) \in \mathscr{L}$ is a candidate approximation to the exact posterior. The goal is to find the best approximation, e.g., the one that satisfies the following optimization problem:

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z}) \in \mathscr{L}}{\operatorname{argmax}} \, \mathrm{KL}[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] \tag{15}$$

Unfortunately, this objective is still not computable because it requires computing the evidence $\log p(x)$ in Equation 14 (that the evidence is hard to compute is why we appeal to approximate inference in the first place). You can see why pretty easily:

$$\begin{aligned} \mathcal{D}_{\mathrm{KL}}[q(z)||p(z|x)] &= \mathbb{E}[\log q(z)] - \mathbb{E}[\log p(z|x)] \\ &= \mathbb{E}[\log q(z)] - \mathbb{E}[\log p(z, x)] + \log p(x) \end{aligned}$$

so we see the dependence on the difficult if not impossible to calculate evidence $p(x)$. Because we cannot compute the KL, we optimize an alternative that is equivalent to the KL up to an added constant:

$$\text{ELBO}(q) = \mathbb{E}[\log p(z, x)] - \mathbb{E}[\log q(z)] \tag{16}$$

This function is called the evidence lower bound (ELBO). The ELBO is the negative KL-divergence minus the log of the evidence, $\log p(x)$ (which is constant with respect to $q(z)$). Importantly, note that maximizing the ELBO is equivalent to minimizing the KL-divergence.

The ELBO also gives some information about the optimal variational distribution. In particular, we can rewrite the ELBO as follows

$$\begin{aligned}\text{ELBO} &= \mathbb{E}[\log p(z)] + \mathbb{E}[\log p(x|z)] - \mathbb{E}[\log q(z)] \\ &= \mathbb{E}[\log p(x|z)] - \text{KL}[q(z)||p(z)]\end{aligned}$$

An interesting and important observation here is that the ELBO is a lower bound on the log evidence. That is, $\log p(x) \geq \text{ELBO}(q), \forall q(z)$. You can see this by noticing that $\log p(x) = \mathcal{D}_{\text{KL}}[q(z)||p(z|x)] + \text{ELBO}(q)$; applying *Jensen's Inequality* (Figure 1) gives $\mathcal{D}_{\text{KL}}(\cdot) \geq 0$. The (lower) bound follows directly from this fact. More below...

Now, suppose we have a vector of latent variables z which has probability density function (pdf) $p(z)$, and that we we can *easily* sample z from $p(z)$. Suppose further that we have a family of functions $f(z; \theta)$ where θ is parameter vector. Then our goal is to optimize θ such that $f(z; \theta)$ produces samples that look like X with high probability (for every $X \in \mathcal{X}$) when z is sampled from $p(z)$. Stated more formally, we wish to maximize the probability of each X in the training set under the *generative process* described by (switching notation a bit)

$$p(x) = \int_z p(\mathbf{x}|\mathbf{z}; \theta)\ p(\mathbf{z})\ dz = \int_z p_\theta(\mathbf{x}|\mathbf{z})\ p(\mathbf{z})\ dz \tag{17}$$

4 Variational Autoencoders

Variational autoencoders attempt to approximately optimize Equation 17.[2] Now, to optimize Equation 17 there are two problems which the VAE must solve. First, the VAE must

[2]Note that VAEs are called autoencoders because the final training objective that derives from this setup does have an encoder and a decoder, and resembles a traditional autoencoder.

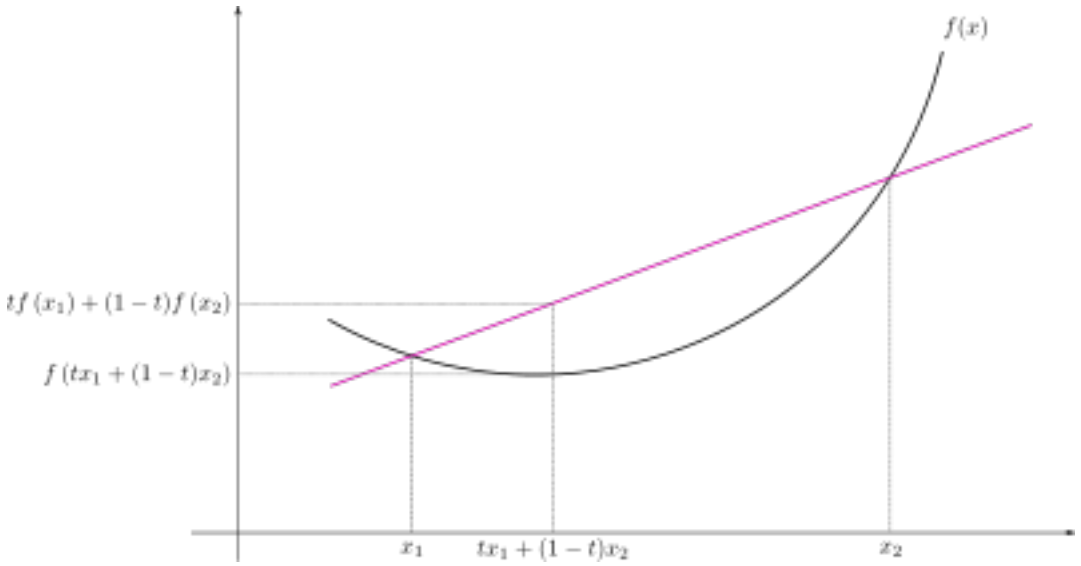


Figure 1: Jensen's Inequality (image courtesy Wikipedia)

describe how to define the latent variables z (that is, decide what information they represent). Second, the VAE must somehow deal with the integral over z (again in Equation 17).

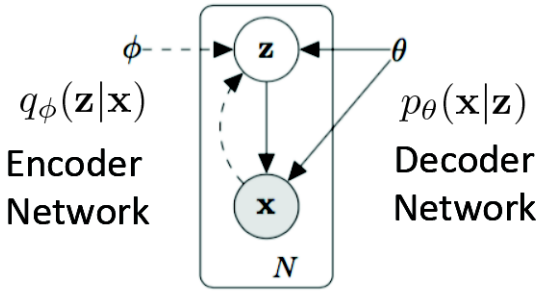
4.1 Choosing the Latent Variables z

When choosing z, we would like to avoid manually deciding what information each dimension of z encodes. We also want to avoid explicitly describing the dependencies (i.e., the structure of $\mathcal{Z}$) between the dimensions of z. VAEs take an novel approach to dealing with this problem: they assume that there is no simple interpretation of the dimensions of z, and instead assert that samples of z can be drawn from a simple distribution, typically $\mathcal{N}(0, I)$ (here again I is the identity matrix). Seems well, impossible.

Note that here $f(z; \theta)$ has been replaced by $p_\theta(\mathbf{x}|\mathbf{z})$ to make the dependence on $\mathbf{x}$ explicit. In VAEs the choice of $p_\theta(\mathbf{x}|\mathbf{z})$ is ofttten a Gaussian such that

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(f(\mathbf{z}; \theta), \sigma^2 * I) \tag{18}$$

That is, $p_\theta(\mathbf{x}|\mathbf{z})$ is a Gaussian distribution with mean $\mu = f(z; \theta)$ and covariance $\Sigma = \sigma^2 * I$, where σ is a scalar hyper-parameter and I is the identity matrix. The important property here is that $p_\theta(\mathbf{x}|\mathbf{z})$ can be efficiently computed.



Minimize: $D_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})]$

Intractable: $p_\theta(\mathbf{z}|\mathbf{x}) = \dfrac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}$

Figure 2: The standard VAE (directed) graphical model

So what is really going on here? The key is to notice that any distribution in d dimensions can be generated by taking a set of d variables that are normally distributed and mapping them through a sufficiently complicated function. Hence, provided powerful function approximators, we can simply learn a function which maps our independent, normally-distributed $\mathbf{z}$ values to whatever latent variables might be needed for the model, and then map those latent variables to $\mathbf{x}$. In fact, recall that $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(f(\mathbf{z};\theta), \sigma^2 * I)$. Now, imagine that $f(z;\theta)$ is a multi-layer neural network, then we can imagine the network using its first few layers to map the normally distributed z's to the latent values with exactly the right properties. So in the case of something like MNIST, it can use later layers to map those latent values to a fully-rendered digit. In general, we don't need to worry about ensuring that the latent structure exists. If such latent structure helps the model accurately reproduce (i.e. maximize the likelihood of) the training set, then the network will learn that structure at some layer.

Recall that our goal is to maximize Equation 17 where $p(\mathbf{z}) = \mathcal{N}(0, I)$. Note that it is actually conceptually straightforward to compute an approximation of $p(\mathbf{x})$. To do this, in theory we can first sample a large number of z values $\{z_1, \cdots, z_n\}$, and then compute $p(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} p(\mathbf{x}|z_i)$. The problem is that if $\mathcal{Z}$ is a high-dimensional space, n may need to be impractically large to get an accurate estimate of $p(\mathbf{x})$; see Appendix 10.2 on the

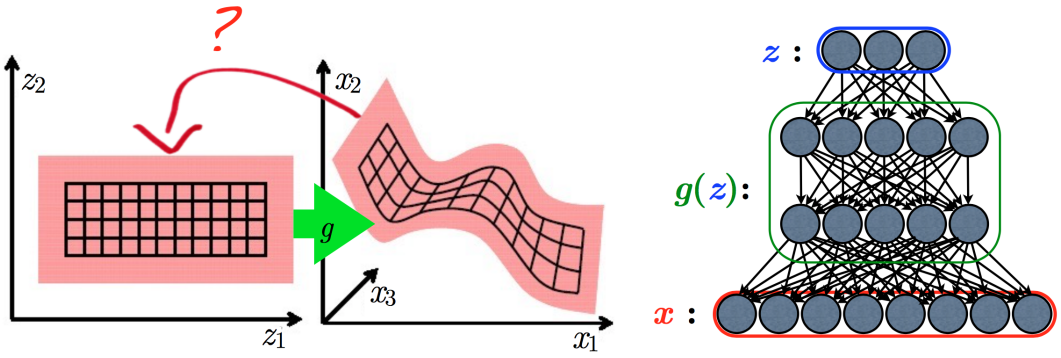


Figure 3: The VAE (directed) Inference/Learning Challenge. Image courtesy `http://videolectures.net/deeplearning2015_courville_autoencoder_extension/`

Ergodic Theorem.

5 VAE Objective Function

So OK, the naive approach requires too many samples from $\mathcal{Z}$ to be practical. However, what we can observe is that for most of the $\mathbf{z}$'s, $p_\theta(\mathbf{x}|\mathbf{z})$ will be close to zero and thus add little to our estimate of $p_\theta(\mathbf{x})$. This leads us to one of the key ideas behind VAEs: VAEs attempt to sample values of $\mathbf{z}$ that are likely to have produced $\mathbf{x}$, and then uses those $\mathbf{z}$'s to compute $p_\theta(\mathbf{x})$.

5.1 So where do the z's come from?

The problem with directly computing the $\mathbf{z}$'s is that posterior $p(\mathbf{z}|\mathbf{x})$ is intractable, but we need it to train the directed model. This situation is depicted in Figure 3. The solution taken by VAEs is introduce an inference machine $q_\phi(\mathbf{z}|\mathbf{x})$ that *learns* to approximate the posterior $p_\theta(\mathbf{z}|\mathbf{x})$. But how?

Suppose that $\mathbf{z}$ is sampled from some arbitrary distribution with pdf $q_\phi(\mathbf{z})$ (not the Gaussian we discussed above). This leads to one of the VAEs key operations: the relationship between $E_{z\sim\ q_\phi(\mathbf{z})}[p_\theta(\mathbf{x}|\mathbf{z})]$ and $p(\mathbf{x})$. But how to do this? One way is to use the Kullback-Leibler divergence $\mathcal{D}_{\mathrm{KL}}$ between $q_\phi(\mathbf{z})$ and $p_\theta(\mathbf{z}|\mathbf{x})$. That is

$$\mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})] = E_{z\sim q_\phi(\mathbf{z})}[\log q_\phi(\mathbf{z}) - \log p_\theta(\mathbf{z}|\mathbf{x})] \tag{19}$$

Applying Bayes' Rule to Equation 19, we get

$$\mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})] = E_{z\sim q_\phi(\mathbf{z})}[\log q_\phi(\mathbf{z}) - \log p_\theta(\mathbf{x}|\mathbf{z}) - \log p_\theta(\mathbf{z})] + \log p(\mathbf{x}) \tag{20}$$

Notice here that $\log p(\mathbf{x})$ can be taken out of the expectation because it doesn't depend on $\mathbf{z}$. Doing a little algebra gives us:

$$\log p(\mathbf{x}) - \mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})] = E_{z\sim q}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathcal{D}_{\mathrm{KL}}[q_\phi(\mathbf{z})||p_\theta(\mathbf{z})] \tag{21}$$

Equation 21 is the basis of the VAE. The left hand side has the quantity we want to maximize: $\log p_\theta(\mathbf{x})$ (plus an error term that we hope is small). The right hand side is something we can optimize via stochastic gradient descent given the right choice of q. That is, we have solved our problem of sampling z by training a distribution q to predict which values of $\mathbf{z}$ are likely to produce $\mathbf{x}$ and ignoring the rest. In particular, on left hand side we are maximizing $\log p(\mathbf{x})$ while simultaneously minimizing $\mathcal{D}_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})]$. Now, $p_\theta(\mathbf{z}|\mathbf{x})$ is not something we can compute analytically: it describes the values of $\mathbf{z}$ that are likely to give rise to a sample like $\mathbf{x}$ under our model in Figure 2. On the other hand, the second term on the left is pulling $q_\phi(\mathbf{z}|\mathbf{x})$ to match $p_\theta(\mathbf{z}|\mathbf{x})$. Assuming we use an arbitrarily high-capacity model for $q_\phi(\mathbf{z}|\mathbf{x})$ then $q_\phi(\mathbf{z}|\mathbf{x})$ will hopefully actually match $p_\theta(\mathbf{z}|\mathbf{x})$, in which case this KL-divergence term will be zero, and we will be directly optimizing $\log p(\mathbf{x})$[3].

But we can go a bit further. We can define a *variational lower bound* $\mathcal{L}$ on the data likelihood such that $p_\theta(\mathbf{x}) \geqslant \mathcal{L}(\theta, \phi, \mathbf{x})$, where

$$\begin{aligned}
\mathcal{L}(\theta, \phi, \mathbf{x}) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)] &(22)\\
&= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z|x)] &(23)\\
&= -D_{\mathrm{KL}}[q_\phi(z|x)||p_\theta(z)] + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] &(24)
\end{aligned}$$

Notes:

- $-D_{\mathrm{KL}}[q_\phi(z|x)||p_\theta(z)]$ is a *regularization* term
- $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$ is a *reconstruction* term
- $\mathbf{x}$ is fixed

[3]Note side effect: we have made the intractable $p_\theta(\mathbf{z}|\mathbf{x})$ tractable: we can just use the computable $q_\phi(\mathbf{z}|\mathbf{x})$ to estimate it.

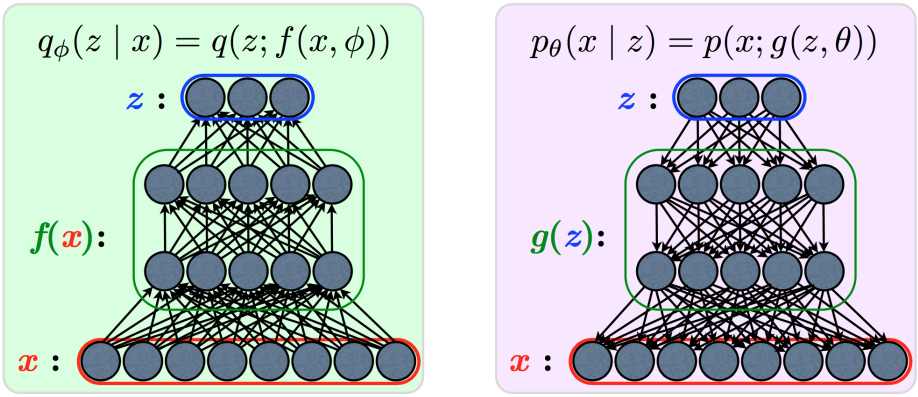


Figure 4: Neural networks for $q_\phi(z|x)$ and $p_\theta(x|z)$

- q_ϕ can be any distribution
- Since we're interested in inferring $p(\mathbf{x})$ it makes sense to construct q using $\mathbf{x}$
- Goal: Find $q_\phi(\mathbf{z}|\mathbf{x})$) so that $\mathcal{D}_{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})]$ is close to zero
- Read $p_\theta(\mathbf{z}|\mathbf{x})$ as "the distribution over the $\mathbf{z}$'s that could have been produced by a sample like $\mathbf{x}$ under the model in Figure 2", which is hopefully a smaller set than all of $p(\mathbf{z})$.

6 VAE Inference Model

The approach taken by VAE's is to introduce an inference model $q_\phi(z|x)$ that learns to approximate the intractable posterior $p_\theta(z|x)$ by optimizing the variational lower bound described above:

$$\mathcal{L}(\theta, \phi, \mathbf{x}) = -D_{\mathrm{KL}}[q_\phi(z|x)||p_\theta(z)] + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] \tag{25}$$

To compute $q_\phi(z|x)$, we parameterize another neural network as shown in Figure 4.

7 One Important "Trick"

First, consider z to be real and $q_\phi(z|x) = \mathcal{N}(z; \mu_z(x), \sigma_z(x))$. Then the *Reparameterization Trick* says we should parametrize z as $z = \mu_z(x) + \sigma_z(x))\epsilon_z$, where $\epsilon_z = \mathcal{N}(0, 1)$. This is depicted in Figure 5. The beautiful thing about the reparameterization trick is that

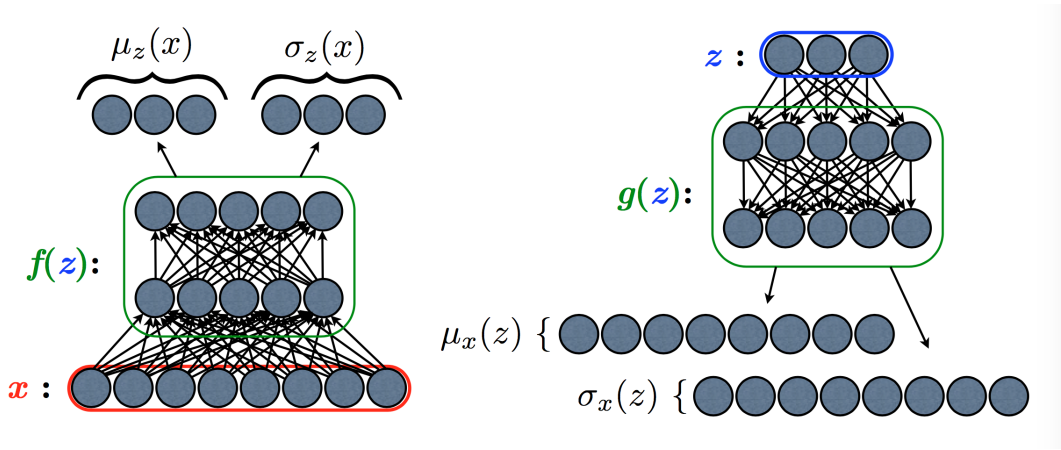


Figure 5: Reparameterization Trick

allows us to use back-propagation, moving the sampling out of the graph into the input layer at training time: see Figure 6. Note that it also to train both the encoder and decoder pathways simultaneously with the objective function shown in Equation 25. This is depicted in Figure 7.

8 A Few Final Thoughts

First, observe that once the VAE is trained, there is no need for the encoder pathway and amazingly, the VAE can generate images (in this case) directly from $\mathcal{N}(0, I)$, as incredible as that may sound. This why the VAE graphical model is frequently drawn as seen in Figure 8.

Finally, Figure 9 shows an implementation of the VAE cost function from [1].

9 Acknowledgements

References

[1] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 12 2013.

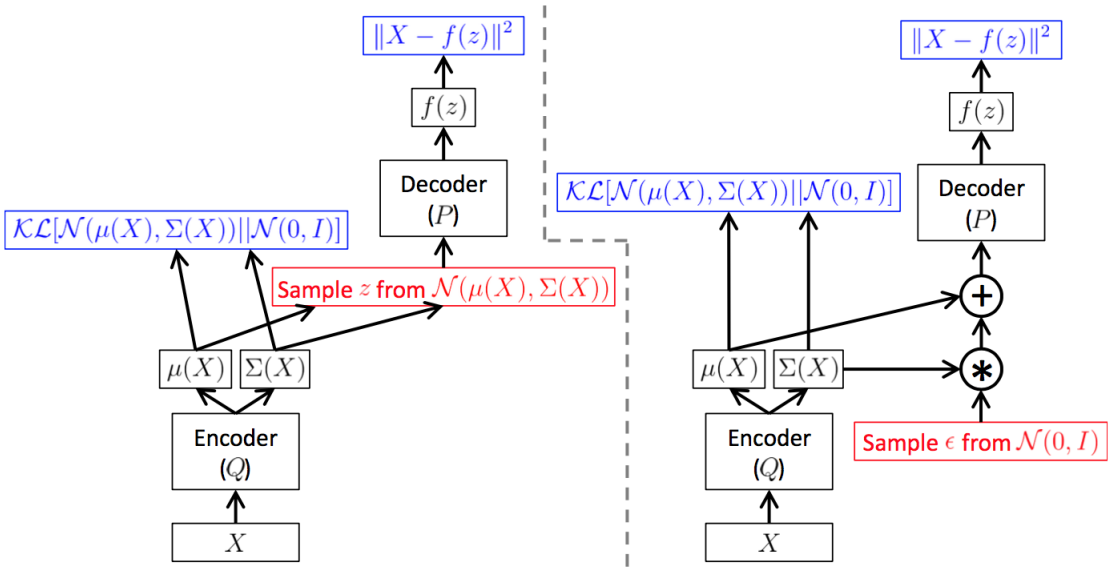


Figure 6: A training-time variational autoencoder implemented as a feed-forward neural network, where $p_\theta(\mathbf{x}|\mathbf{z})$ is Gaussian. The network on the left is without the *reparameterization trick*, and network on the right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but back propagation can be applied only to the right network. Image courtesy `https://arxiv.org/abs/1606.05908`

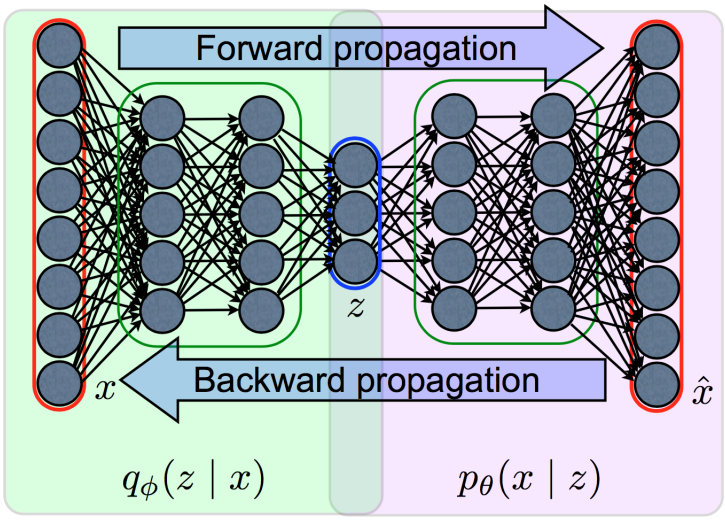


Figure 7: Training the VAE via Backpropagation

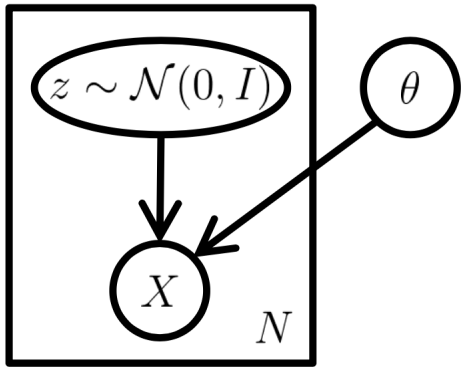


Figure 8: VAE Graphical Model With No Encoder Pathway

10 Appendix

10.1 Strong Law of Large Numbers

Let $X_1, X_2, \ldots, X_M$ be a sequence of **independent** and **identically distributed** random variables, each having a finite mean $\mu_i = E[X_i]$.

Then with probability 1

$$\frac{1}{M}\sum_{i=1}^{M} X_i \to E[X] \tag{26}$$

as $M \to \infty$.

10.2 Ergodic Theorem

Let $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}$ be M samples from a Markov chain that is *aperiodic*, *irreducible*, and *positive recurrent*[4], and $E[g(\theta)] < \infty$.

Then with probability 1

$$\frac{1}{M}\sum_{i=1}^{M} g(\theta_i) \to E[g(\theta)] = \int_{\Theta} g(\theta)\, \pi(\theta)\, d\theta \tag{27}$$

as $M \to \infty$ and where π is the stationary distribution of the Markov chain.

[4] In this case, the chain is said to be *ergodic*.

```
#
#       _optimize_variational_lower_bound
#
#       Here we use the variational lower bound as a proxy for the
#       cost function. That is, we want to optimize (variational)
#       lower bound on the marginal likelihood, which is composed of
#       the reconstruction loss and the KL divergence of the
#       approximate (variational) from the true posterior.
#
#       See Kingma and Welling, "Auto-Encoding Variational Bayes",
#       https://arxiv.org/pdf/1312.6114v10.pdf for more detail.
#
#       In practice, the cost  here is composed of two terms:
#
#       (1)     The reconstruction loss
#
#               This is the negative log probability of the input
#               under the reconstructed Bernoulli distribution induced
#               by the decoder in the data space. This component is
#               computed in reconstr_loss. See appendix C.1 of
#               https://arxiv.org/pdf/1312.6114v10.pdf for the details
#               of the Bernoulli MLP decoder
#
#       (2)     The latent loss
#
#               This is defined to be the Kullback Leibler divergence
#               between the distribution in latent space induced by
#               the encoder on the data and some prior. It acts as a
#               regularizer. See appendix B of
#               https://arxiv.org/pdf/1312.6114v10.pdf for the
#               solution of -DKL(q(z)||p(z)) in the Gaussian case
#
#       Note finally that in the below we add 1e-10 to avoid evaluatio of log(0.0)
#
    def _optimize_variational_lower_bound(self):
        reconstr_loss  = -tf.reduce_sum(self.x * tf.log(1e-10 + self.x_reconstr_mean)
                           + (1-self.x) * tf.log(1e-10 + 1 - self.x_reconstr_mean), 1)
        latent_loss    = -0.5 * tf.reduce_sum(1 + self.z_log_sigma_sq
                                              - tf.square(self.z_mean)
                                              - tf.exp(self.z_log_sigma_sq), 1)
        self.cost      = tf.reduce_mean(reconstr_loss + latent_loss)         # average over batch
        self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate).minimize(self.cost)
```

Figure 9: Cost Function from "Auto-encoding Variational Bayes"