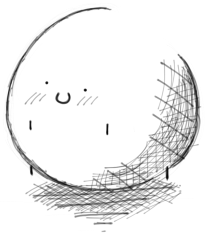


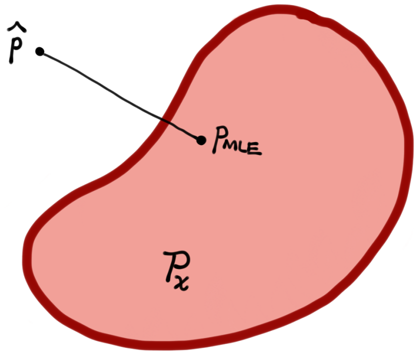
Rui Shu

14 MAR 2018

DENSITY ESTIMATION: VARIATIONAL AUTOENCODERS

One of the most popular models for density estimation is the Variational Autoencoder. It is a model that I have spent a considerable amount of time working with, so I want to give it an especially in-depth treatment. In this post, we shall construct the VAE ground-up from statistical principles. We will go over the key concepts that underpin the VAE model: maximum likelihood estimation, latent variable models, variational inference, and parameter optimization. Only at the very end will we introduce neural networks as a means of parameterizing conditional distributions.

Maximum Likelihood Estimation



Maximum likelihood estimation finds the distribution $p \in \mathcal{P}_x$ that best approximates $\hat{p}$ under the Kullback-Leibler divergence. The selected p_{MLE} is sometimes interpreted as the projection of $\hat{p}$ onto $\mathcal{P}_x$.

We are interested in estimating a true distribution $p^*(x)$. To do so, we first introduce a set of set of candidate distributions $\mathcal{P}_x$. Each $p \in \mathcal{P}_x$ is distribution that defines a density over x. Next, although we do not have access to $p^*(x)$, we do have access to finite samples from it. We denote uniform sampling from this finite dataset as $\hat{p}(x)$. The goal of maximum likelihood estimation is to find the best $p \in \mathcal{P}_x$ to approximate $\hat{p}(x)$ as measured by the Kullback-Leibler divergence,

$$\begin{aligned}\min_{p \in \mathcal{P}_x} D(\hat{p} \parallel p_g) &= \min_{p \in \mathcal{P}_x} \mathbb{E}_{\hat{p}(x)} \left[\ln \frac{\hat{p}(x)}{p(x)} \right] \\ &\equiv \max_{p \in \mathcal{P}_x} \mathbb{E}_{\hat{p}(x)} \left[\ln p(x) \right].\end{aligned}$$

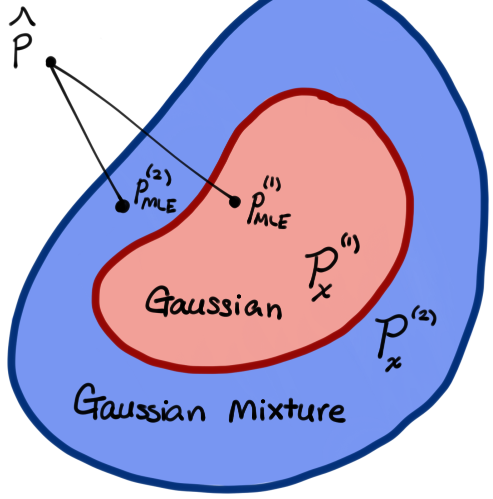
Note that minimizing the KL divergence is equivalent to maximizing the log likelihood.

Latent Variable Models

We call $\mathcal{P}_x$ the hypothesis space of generative models. Given the maximum likelihood problem formulation, we wish for our hypothesis space $\mathcal{P}_x$ to exhibit at least two desirable traits

1. Any hypothesis $p \in \mathcal{P}_x$ must have a tractable-to-compute density, since our objective requires that we compute $p(x)$.
2. $\mathcal{P}_x$ should be expressive. Ideally, we want p^* to be contained within the distribution family $\mathcal{P}_x$. Since we don't know what p^* is a priori, it is desirable to make $\mathcal{P}_x$ sufficiently flexible—but not so flexible that it includes $\hat{p}$ (i.e. memorization).

Unfortunately, the first and second conditions are typically at odds with each other. Generally speaking, very few distributions have densities that are computationally tractable. This puts a severe constraint on the flexibility of $\mathcal{P}_x$, restricting it to only the set of "simple" distributions.



A Gaussian Mixture family $\mathcal{P}_x^{(2)}$ is more expressive than a Gaussian family $\mathcal{P}_x^{(1)}$ and is thus better at approximating the empirical distribution. Formally, since $\mathcal{P}_x^{(1)} \subseteq \mathcal{P}_x^{(2)}$, it follows that $D(\hat{p} \parallel p^{(1)}_{\text{MLE}}) \geq D(\hat{p} \parallel p^{(2)}_{\text{MLE}})$, where $p^{(1)}_{\text{MLE}}$ is the projection onto the Gaussian family, and $p^{(2)}_{\text{MLE}}$ is the the projection onto the Gaussian Mixture family.

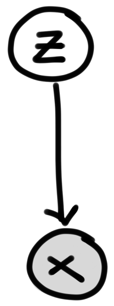
One of the many motivations of a latent variable model is to introduce latent variables so that the joint distribution can be defined as a product of simple distributions. Take, for example, the well-known distribution family: a mixture of k Gaussians. We introduce the discrete latent variable z which can take on values in $\{1, \ldots, k\}$ according to the probability vector π and a conditional Gaussian distribution parameterized by $\{\mu_z, \Sigma_z\}_{z=1}^k$. Note that the joint distribution $p(x, z)$ has a tractable density computed as the product of a categorical and a Gaussian density

$$p(z)p(x \mid z) = \mathrm{Cat}(z \mid \pi) \cdot \mathcal{N}(x \mid \mu_z, \Sigma_z) = \pi_z \cdot \mathcal{N}(x \mid \mu_z, \Sigma_z),$$

and for which the density over x is

$$p(x) = \sum_{z=1}^{k} \pi_z \cdot \mathcal{N}(x \mid \mu_z, \Sigma_z).$$

Despite the decomposition into two fairly simple distributions, the density $p(x)$ is achieved through the marginalization of the latent variable z, yielding a distribution family (the Gaussian mixture family) over x that is strictly more expressive than the Gaussian family. As such, one can think of latent variable models as leveraging auxiliary variables to express more complex distributions.



A graphical representation of our latent variable model, where x is our observed variable and z is our latent variable. Training this model via maximum likelihood entails identifying the best distribution $p(z) \in \mathcal{P}_z$ and conditional distribution $p(x \mid z) \in \mathcal{P}_{x|z}$ such that marginal $p(x)$ minimizes the KL-divergence to the empirical $\hat{p}(x)$.

Given the flexibility of latent variable models, we shall instead train a latent variable model. To do so, we construct $\mathcal{P}_{x,z} = \mathcal{P}_z \times \mathcal{P}_{x|z}$, where $\mathcal{P}_z$ is a set of simple distributions over z and $\mathcal{P}_{x|z}$ is a set of simple conditional distributions over x (given z). As such, each $(p_z, p_{x|z}) \in \mathcal{P}_{x,z}$ defines a density jointly over the observed variable x and the latent variable z. For notational simplicity, we shall in the remaining text refer to $p \in \mathcal{P}$ as a hypothesis that defines the joint distribution, and denote $p(z) = p_z(z)$, $p(x \mid z) = p_{x|z}(x \mid z)$.

Intractable Marginal and the Variational Principle

Although we have deliberately chosen $\mathcal{P}$ such that the joint distribution has a tractable density, note that the computation of $p(x)$ requires marginalization of the latent variable z. If z is a continuous variable, then marginalization takes the form

$$\ln p(x) = \ln \int p(z)p(x \mid z)dz = \mathbb{E}_{p(z)}\left[p(x \mid z)\right]$$

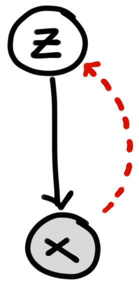
This marginalization is potentially very difficult to compute. To get around this issue, we rely on the variational principle: the technique of converting a difficult computation problem into an optimization problem. Fortunately it is easy to construct a lower bound on the likelihood $p(x)$ by introducing a family of conditional distributions $\mathcal{Q}$. The family $\mathcal{Q}$ is called the variational family and its elements $q(z \mid x) \in \mathcal{Q}$ define a distribution over z conditioned on x. Using Jensen's inequality, note that for any $q \in Q$,

$$\ln p(x) = \ln \int \frac{q(z \mid x)}{q(z \mid x)} p(x, z) dz \overset{(a)}{\geq} \int q(z \mid x) \ln \frac{p(x, z)}{q(z \mid x)} dz = \mathbb{E}_{q(z|x)} \ln \frac{p(x, z)}{q(z \mid x)},$$

where (a) denotes the application of Jensen's Inequality. This lower bound is called the variational lower bound. Rather than computing $\ln p(x)$ directly by marginalization, we instead optimize the variational lower bound over $q \in \mathcal{Q}$. To provide an intuition for what optimizing over $q \in \mathcal{Q}$ does, note that there is an alternative formulation of this lower bound. By applying to chain rule $p(x, z) = p(x)p(z \mid x)$, we see that optimizing the variational lower bound is equivalent to

$$\max_{q \in \mathcal{Q}} \mathbb{E}_{q(z|x)} \ln \frac{p(x, z)}{q(z \mid x)} = \ln p(x) - \min_{q \in \mathcal{Q}} D(q(z \mid x) \parallel p(z \mid x)).$$

This shows that optimizing the lower bound is equivalent to finding $q \in \mathcal{Q}$ that best approximates the posterior $p(z \mid x)$. This exposes the interesting connection between marginalization and posterior inference, and is also why maximizing the variational lower bound is called performing variational inference.



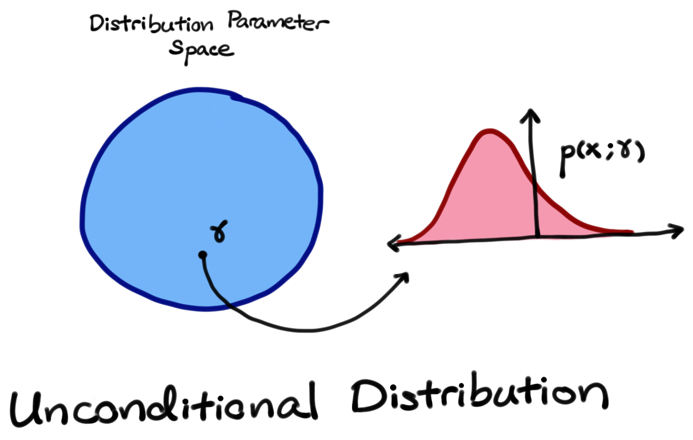
The variational objective suggests that rather than maximizing the log likelihood directly, we instead maximize its variational lower bound, which entails estimating the posterior $p(z \mid x)$ with the variational approximation $q(z \mid x)$.

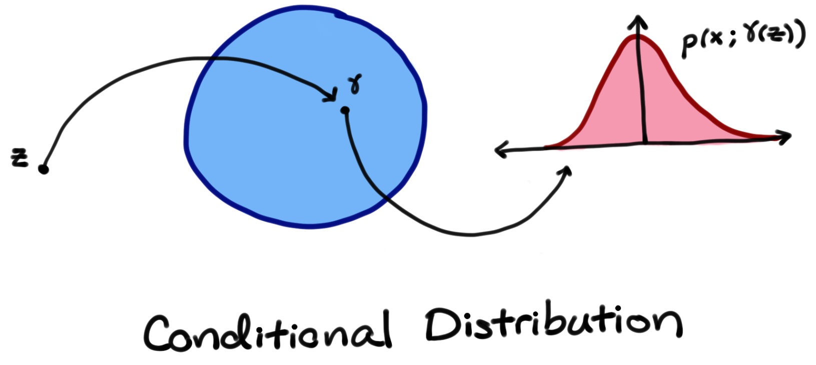
Rather than maximizing the log likelihood directly, we now solve the optimization problem

$$\max_{p \in \mathcal{P}, q \in \mathcal{Q}} \mathbb{E}_{\hat{p}(x)} \mathbb{E}_{q(z|x)} \ln \frac{p(x, z)}{q(z \mid x)}.$$

Note that in doing so, we not only learn to approximate $p^*(x)$, but we also simultaneously figure out how to best approximate the posterior of our latent variable model $p(z)p(x \mid z)$.

Parametric Models for Distributions





We consider the case of unconditional and conditional parametric distributions, where the distribution $p(x\,;\gamma)$ is completely defined by parameters γ. To represent a conditional distribution $p(x \mid z)$, simply construct the distribution $p(x\,;\gamma(z))$ where γ is a function of z.

So far, we've been talking about distributions and conditional distributions in the abstract. To implement variational autoencoders in practice, we will instantiate these abstractions with actual distributions. For the sake of implicity, we shall restriction $\mathcal{P}_z$ to contain only the unit Gaussian distribution

$$\mathcal{P}_z = \left\{p \mid p(z) = \mathcal{N}(z \mid \mathbf{0}, \mathbf{I})\right\}.$$

Note that a diagonal Gaussian distribution is always defined by its mean and standard deviation parameters $\gamma = (\mu, \sigma)$. This provides a means of constructing a conditional diagonal Gaussian distribution: simply make μ and σ functions of z. In other words, we can build the observation model $p(x \mid z)$ as

$$p(x \mid z) = \mathcal{N}(x \mid \mu_\theta(z), \mathrm{diag}(\sigma^2_\theta(z))).$$

Now, μ_θ and σ_θ with function parameters θ. If μ_θ and σ_θ are neural networks, then θ denotes the weight of the neural network. Similarly, we can build the inference model as

$$q(z \mid x) = \mathcal{N}(z \mid \mu_\phi(x), \mathrm{diag}(\sigma^2_\phi(x))),$$

whose neural network parameters are ϕ. Since we parameterized the conditional distributions of our latent variable model with neural networks, we call our model a deep latent variable model.

Note that the conditional form of *any* parametric distribution can be constructed by converting the distribution parameters γ into functions $\gamma(z)$. As such, one does not need to use a conditional Gaussian distribution for their observation or inference model. This post, however, shall only consider conditional Gaussian models.

Reparameterization Trick

To train our deep latent variable model, our original optimization problem

$$\max_{p \in \mathcal{P}, q \in \mathcal{Q}} \mathbb{E}_{\hat{p}(x)} \mathbb{E}_{q(z|x)} \ln \frac{p(x, z)}{q(z \mid x)}.$$

takes on a more concrete form

$$\max_{\theta, \phi} \mathbb{E}_{\hat{p}(x)} \mathbb{E}_{q_\phi(z|x)} \ln \frac{p(z) p_\theta(x \mid z)}{q_\phi(z \mid x)}.$$

To optimize this objection, we shall—in the tradition of deep learning models—rely on stochastic gradient descent. This presents a little bit of a challenge, however, since we will need to compute the gradient

$$\nabla_\phi \mathbb{E}_{q_\phi(z|x)} \ln \frac{p(z) p_\theta(x \mid z)}{q_\phi(z \mid x)}.$$

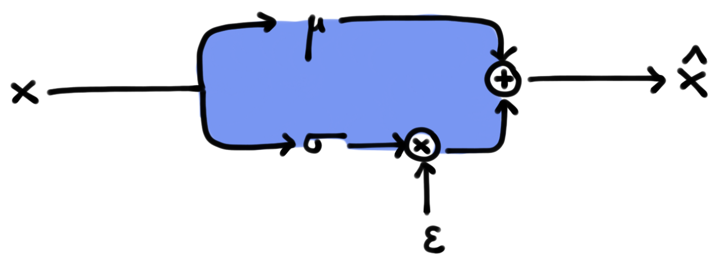
Since the expectation depends on ϕ, we cannot push the gradient through the expectation. To resolve this issue, we need a way to sample from $q_\phi(z \mid x)$ but, at the same time, not have the expectation depend on $q_\phi(z \mid x)$. To do so, we rely on the trick of transformation-based sampling, where we sample ϵ from a fixed distribution but come up with a transformation function T such that sampling $T(\epsilon \,;\, \phi, x)$ is equivalent to sampling from $q_\phi(z \mid x)$. In other words, we need to find an appropriate choice of $p(\epsilon)$ and $T(\epsilon)$ such that

$$\nabla_\phi \mathbb{E}_{q_\phi(z|x)} \ln \frac{p(z) p_\theta(x \mid z)}{q_\phi(z \mid x)} = \nabla_\phi \mathbb{E}_{p(\epsilon)} \ln \frac{p(T(\epsilon)) p_\theta(x \mid T(\epsilon))}{q_\phi(T(\epsilon) \mid x)},$$

where the dependency of T on (ϕ, x) is omitted for notational simplicity. Fortunately, when $q_\phi(z \mid x)$ is a conditional diagonal Gaussian model, a simple choice of $p(\epsilon)$ and T exists

$$\begin{aligned} p(\epsilon) &= \mathcal{N}(\epsilon \mid \mathbf{0}, \mathbf{I}) \\ T(\epsilon \,;\, \phi, x) &= \mu_\phi(x) + \epsilon \odot \sigma_\phi(x). \end{aligned}$$

For non-Gaussian conditional models, one will need to work out an appropriate choice of $(p(\epsilon), T)$ from scratch.



The computational graph for the training of our deep latent variable model exposes its connection to autoencoders.

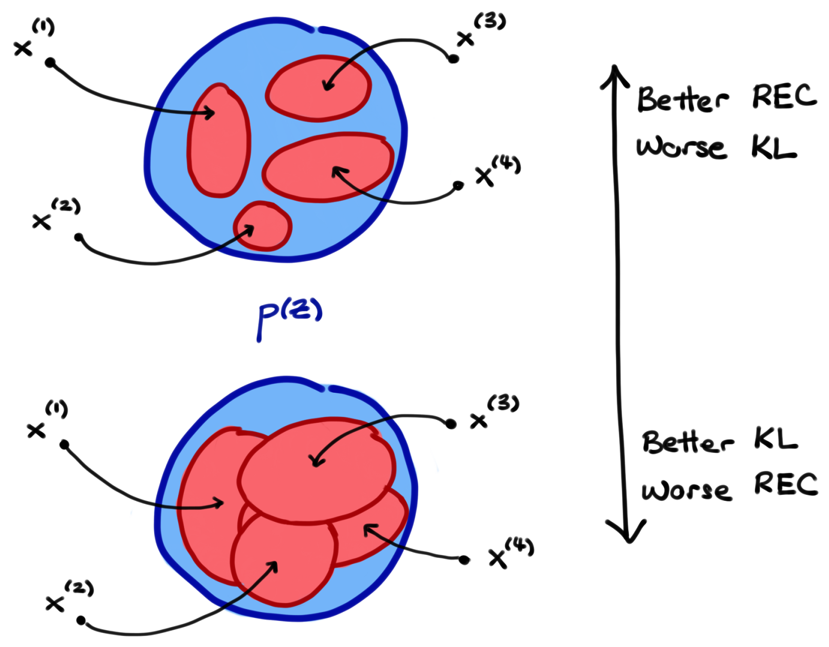
The transformation-based sampling also exposes the connection between autoencoders and deep latent variable models optimized via the variational lower bound. The forward pass of the computational graph during training includes the path

1. Compute $\mu_\phi(x)$ and $\sigma_\phi(x)$.
2. Sample $z = \mu_\phi(x) + \epsilon \odot \sigma_\phi(x)$, where z can be interpreted as the stochastic embedding.
3. Compute $\mu_\theta(z)$ and $\sigma_\theta(z)$, where $\mu_\theta(x)$ can be interpreted as $\hat{x}$.

This motivates our final reformulation of the variational objective. By applying to chain rule $p_\theta(x, z) = p(z)p_\theta(x \mid z)$, we see that the variational lower bound is equivalent to

$$\mathbb{E}_{q_\phi(z|x)} \ln \frac{p_\theta(x, z)}{q_\phi(z \mid x)} = \mathbb{E}_{q_\phi(z|x)} \ln p_\theta(x \mid z) - D(q_\phi(z \mid x) \parallel p(z)).$$

The first term $\mathbb{E}_{q_\phi(z|x)} \ln p_\theta(x \mid z)$ can be interpreted as the autoencoding (reconstruction) objective. The second term regularizes $q_\phi(z \mid x)$ to be close to the prior $p(z)$. This introduces a tug-of-war effect: to get better reconstructions, the embedding means $\mu_\phi(x)$ are pushed far away from each other and embedding standard deviations $\sigma_\phi(x)$ are pulled toward zero; to get smaller $D(q_\phi(z \mid x) \parallel p(z))$, the embedding means are pulled toward zero and the embedding standard deviations are pulled toward one.



An idealized representation of four samples being stochastically encoded into the space of z. REC denotes the reconstruction cost, while KL denotes the KL-divergence to the prior. When there is little overlap over the stochastic embeddings, the reconstruction cost is low but the KL-divergence is high. When the KL-divergence is low, there is too much overlap between the embeddings and thus reconstruction cost is high.

This autoencoding reformulation of our deep latent variable model objective is what inspires the model's name as the Variational Autoencoder. We stress, however, that there is value in the Jensen's Inequality formulation

$$\mathbb{E}_{q_\phi(z|x)} \ln \frac{p_\theta(x, z)}{q_\phi(z \mid x)},$$

which informs how to construct hierarhical deep latent variable models, as well as the posterior divergence formulation

$$\ln p_\theta(x) - D(q_\phi(z \mid x) \parallel p_\theta(z \mid x)),$$

which exposes the connection between the variational distribution and the generative model's true posterior.

Implementation: Weird Tricks and Their Theoretical Underpinnings

There is an abundance of variational autoencoder implementations on the web, so I won't belabor the details of implementing a VAE too much. If you want to see my implementation of a standard VAE in its entirely, simply check out this Github repository. For the sake of completeness, I will highlight certain portions of the code below to indicate how it relates to the theory of VAE that we have developed thus far. Most importantly, I will identify certain tricks that VAE practitioners use to ensure the numerical stability of their model and discuss the theoretical significance of these tricks.

First and foremost, let's construct the neural networks that parameterize our conditional distributions $p_\theta(x \mid z)$ and $q_\phi(z \mid x)$, which I refer to respectively as the generator and encoder. Here, I use a pretty straightforward parameterization: the encoder uses a series of convolutional layers, and the generator a series of transposed convolutional layers.

```
def encoder(x, y=None, phase=False, scope='enc', reuse=None, internal_update=Fal
    with tf.variable_scope(scope, reuse=reuse):
        with arg_scope([conv2d, dense], bn=True, phase=phase, activation=leaky_re
             arg_scope([batch_norm], internal_update=internal_update):

            x = conv2d(x, 32, 3, 2)
            x = conv2d(x, 64, 3, 2)
            x = conv2d(x, 128, 3, 2)
            x = dense(x, 1024)

            m = dense(x, args.Z, activation=None)
            v = dense(x, args.Z, activation=tf.nn.softplus) + 1e-5
            z = gaussian_sample(m, v)

    return z, (m, v)

def generator(z, y=None, phase=False, scope='gen', reuse=None, internal_update=Fa
    with tf.variable_scope(scope, reuse=reuse):
        with arg_scope([dense, conv2d_transpose], bn=True, phase=phase, activati
             arg_scope([batch_norm], internal_update=internal_update):

            if y is not None:
                z = tf.concat([z, y], 1)

            z = dense(z, 4 * 4 * 512)
            z = tf.reshape(z, [-1, 4, 4, 512])
            z = conv2d_transpose(z, 128, 5, 2)
            z = conv2d_transpose(z, 64, 5, 2)
            x = conv2d_transpose(z, 3, 5, 2, bn=False, activation=tf.nn.tanh)

    return x
```

Lower bounding the Inference Model Variance

The encoder then uses two separate heads to propose the mean and variance parameters of the conditional distribution. Here, I compute the variance directly, using a softplus activation. A popular alternative is to compute the standard deviation, using the exponential activation. Notice that I add a fudge factor `1e-5` to the variance. This fudge factor is there to impose a lower bound on the variance achievable by $q_\phi(z \mid x)$ and is introduced for numerical stability: if $q_\phi(z \mid x)$ is allowed to have arbitrarily small variance, the density tends toward infinity. Not imposing a lower bound on the variance of the inference model is a most common cause of `nan`'s in VAE implementations.

Lower bounding the Generative Model Variance

The next thing to note is that our generator only returns the mean parameter $\mu_\theta(z)$ but not the variance parameter $\sigma^2_\theta(z)$ of our observation model $p_\theta(x \mid z) = \mathcal{N}(x \mid \mu_\theta(z), \text{diag}(\sigma^2_\theta(z)))$. This is because we will simply set to the variance globally to 1. One is is rightfully concerned and should wonder why we deliberately chose *not* to learn the variance parameter of our obseravtion. Setting the variance is 1 is especially incredible; if our dataset consists of images with normalized pixel intensities in $[-1, 1]$, using a variance of 1 adds a remarkable amount of noise when sampling from $p_\theta(x \mid z)$. The dirty secret when using Gaussian observation model VAEs is that no one actually samples from $p_\theta(x \mid z)$, but instead directly report $\mu_\theta(z)$ as "samples" from the generator. This is by far the dirtiest and most prevalent trick that practitioners employ when using Gaussian observation model VAEs. There is, to some extent, good theoretical justification for this practice. First, one should note that for the same reason as mentioned in the previous paragraph, there is value in lower bounding the variance of the observation model. This is especially critical for the observation model as the variance directly governs the *weighting* of the ℓ_2-reconstruction loss, since

$$\ln p_\theta(x \mid z) = -\frac{1}{2\sigma^2_\theta(z)}\|x - \mu_\theta(z)\|^2 - \frac{1}{2}\ln 2\pi\sigma^2_\theta(z).$$

If your model has sufficient capacity such that there exists (θ, ϕ) for which $\mu_\theta(z)$ provides a sufficiently good reconstruction of x, then $-\ln \sigma^2_\theta(z)$ will encourage the variance to go close to zero first before $\frac{1}{\sigma^2_\theta(z)}$ catches up. Consequently, the variational objective will shrink the variance of the Gaussian observation model toward zero aggressively. A more general (and mathematically-inclined) interpretation is that the density of finite samples under the Lebesgue measure is undefined; thus, if there exists θ such that the image of μ_θ contains the empirical samples, then it is possible to construct a sequence of (θ_n) for which the log likelihood $\ln p_\theta(x)$ tends toward infinity. For carefully-chosen μ_ϕ (under some assumptions about the expressivity of the inference model), the variational lower bound can tend toward infinity as well. Motivated by this consideration, the de facto practice when using Gaussian observation models is to set the decoder variance as a global hyperparameter. And since the variance of factorized Gaussian models interpreted as noising the samples, practitioners often choose to omit actual sampling of $p_\theta(x \mid z)$ when training VAE on image datasets—all so their generated images will look prettier.

Reparameterized Sampling

Finally, note the use of the function `gaussian_sample`. This is basically a thin wrapper around `tf.random_normal`, which itself implements the reparameterization trick by transforming a unit Gaussian according to the provided mean and standard deviation parameters.

```
def gaussian_sample(mean, var, scope=None):
    with tf.variable_scope(scope, 'gaussian_sample'):
        sample = tf.random_normal(tf.shape(mean), mean, tf.sqrt(var))
        sample.set_shape(mean.get_shape())
        return sample
```

Computing and Optimizing the Variational Objective

Now that we've constructed our encoder and generator, implementing the VAE computational graph is pretty simple. We first take a sample `T.trg_x` and push it through the encoder. The encoder returns the stochastic embedding z as well as the parameters of the underlying distribution from which z was drawn. Next we feed z to the decoder to compute $\hat{x}$.

```
# Inference
z, z_post = nn.encoder(T.trg_x, phase=True, internal_update=True)

# Generation
x = nn.generator(z, phase=True, internal_update=True)

# Loss
z_prior = (0., 1.)
loss_kl = tf.reduce_mean(log_normal(z, *z_post) - log_normal(z, *z_prior))
loss_rec = tf.reduce_mean(reduce_l2_loss(x - x_prior, axis=[1,2,3]))
loss_gen = loss_kl + loss_rec
```

Using the Jensen's Inequality formulation of the (negative) variational objective, it is easy to see that we simply need to minimize the quantity

$$\mathbb{E}_{q_\phi(z|x)} - \ln p_\theta(x \mid z) + \ln q_\phi(z \mid x) - \ln p_\theta(z),$$

which translates directly to sampling a stochastic embedding and then computing

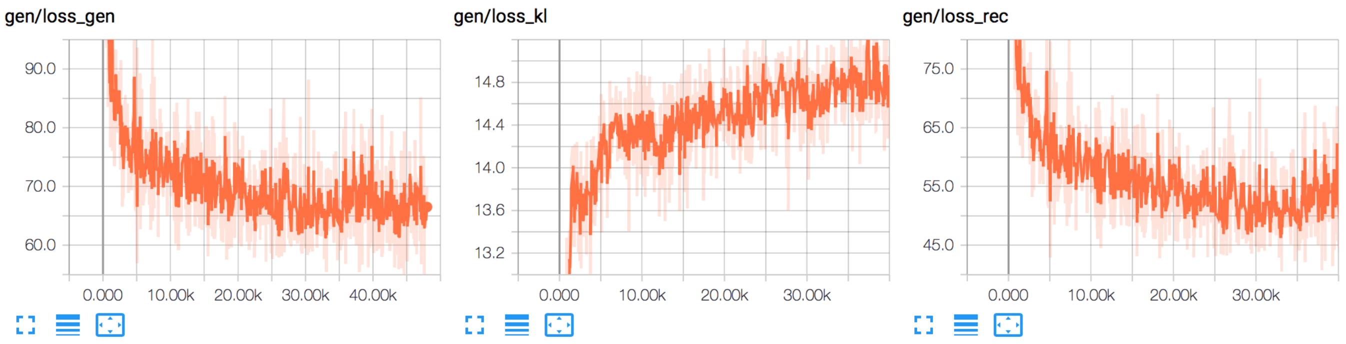
```
rec = -log_normal(T.trg_x, x, 1)
kl = log_normal(z, z_mean, z_variance) - log_normal(z, 0, 1)
```

Since the variance for the observation model is fixed to 1, the reconstruction loss is simply the ℓ_2-loss $0.5 * \|x - \mu_\theta(z)\|^2$. Once set up, everything is trained with stochastic gradient descent or any of its variants. In my case, I've chosen to use the Adam optimizer.

```
var_main = tf.get_collection('trainable_variables', 'gen/')
var_main += tf.get_collection('trainable_variables', 'enc/')
loss_main = loss_gen
train_main = tf.train.AdamOptimizer(args.lr, 0.5).minimize(loss_main, var_list=va
```

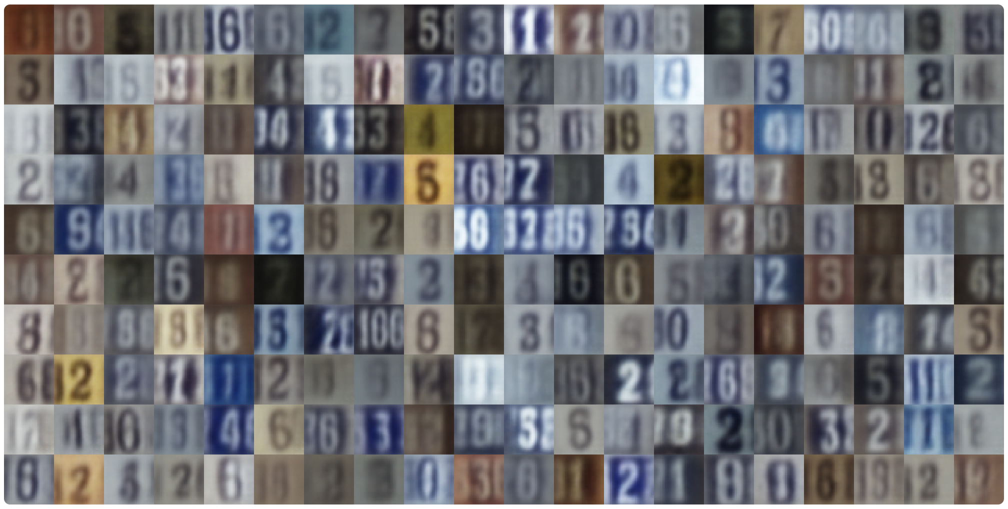
Results

With everything set up, we can now test our VAE on a dataset. In lieu of MNIST, I thought it'd be more interesting to test VAE on the somewhat more challenging SVHN dataset. After about 50000 mini-batch updates, the resulting loss curves are as follows.



Loss curves when training a VAE on the SVHN dataset.

And the sampled images (more specifically, samples of $\mu_\theta(z)$) are starting to look reasonable.


Random samples from our VAE.

Conclusion

This post was designed to provide an in-depth look at the theory and practice of variational autoencoders. Many of the points I've discussed here are points that are also touched on by Carl Doersch in his Variational Autoencoder Tutorial, although we differ somewhat in our choice of presentation and emphasis. In particular, this post takes considerable care in separating the theory of learning latent variable models from the choice of parameterizing such models with deep neural networks. In doing so, I hope the reader will gain a better appreciation for the underlying mathematics of maximum likelihood and variational inference, as well as the elegance with which the Variational Autoencoder marries these ideas with deep learning techniques.

For those who are interested in knowing more about variational autoencoders, I am currently working on an even more research-oriented tutorial on variational autoencoders. There are some concepts that I've glossed over in this post which I think deserves extra careful treatment. I'll try to have the tutorial ready sometime after the NIPS deadline c:

CODE ON GITHUB

End of post

Alex Telfar • 3 years ago

You mentioned density estimation in this post. How do you use the VAE to actually estimate p(x)? It isnt clear to me...

2 | • Reply • Share ›

Rui Mod Alex Telfar • 3 years ago • edited

VAEs allow fast computation of the joint distribution p_theta(x, z). However, computing the marginal p_theta(x) exactly is often intractable. If we are willing to pay a computational price, we can get pretty tight lower bounds of ln p(x) using annealed importance sampling (AIS). AIS is generally accepted as a reliable approximation of ln p(x) for VAEs.

1 | • Reply • Share ›

istvan ➦ Rui • 3 years ago

I'm still wondering about the motivation and what gets accomplished.
You wrote "This marginalization is potentially very difficult to compute. To get around this issue, [...]". But then the VAE doesn't really get around the issue, since you're still referring to this marginalization problem in your comment. But then how is this a density estimation method?

1 ^ | ⌄ • Reply • Share ›

Rui Mod ➦ istvan • 3 years ago

That's a fair point. If the goal is to learn a model that gives exact densities, then one should think twice about using VAEs (or any latent var model with complex likelihood functions).

We seem to disagree on what counts as density estimation. To the extent that VAE is an explicit likelihood-based model and for which a well-defined (albeit potentially intractable) procedure exists for approximating p(x), I am comfortable classifying it as a density estimation model. For the exact same reason, I'm also inclined to classify MRFs/CRFs as density estimation models.

^ | ⌄ 1 • Reply • Share ›

youni • 2 years ago • edited

Thanks for your great post! I really enjoyed reading it. I have two questions.
1. It is known that marginal p(x) = \int p(z|x)p(z) dz is often intractable, and we resort to maximizing ELBO with q(x|z). However, what if we just approximate p(x) = \int p(x|z) p(z) dz with MC sampling, as is done in approximating the expectation in ELBO? And maximize this MC approximated p(x) to learn theta?
2. VAE is a density estimation model, based on maximum likelihood estimation. Then, what is the connection of VAE to the Bayesian framework? (i.e. the meaning of the name, variational "Bayes"?)

1 ^ | ⌄ • Reply • Share ›

Rui Mod ➦ youni • 2 years ago • edited

1a. Directly optimizing in probability mass/density space for high-dimensional data can be practically challenging. These numbers can get extremely large/small. That's why in general we work with log-probability mass/density.

1b. While MC estimation of p(x) will be unbiased, it tends to suffer from extremely high variance. This high-variance in fact has a strong correspondence to why the log of the unbiased estimator becomes a highly loose lower bound of log p(x).

2. I think the connection is superficial. Of course VAEs can be applied to Bayesian problems (just treat your parameters as the latent variables). The use of terminology like "variational bayes", "evidence lower bound" and "posterior inference" comes from these methods originally being developed for inference in Bayesian settings. But in general, I would not personally characterize VAEs as bayesian.

1 ^ | ⌄ • Reply • Share ›

William Gan ➦ youni • 2 years ago

I was confused with 1 too. I think because q(z|x) and p(z) are Gaussians with fixed means and variances, you can compute E_q(z|x) ln q(z|x) and E_q(z|x) ln p(z) analytically. That's what it seems like they're doing in Appendix B of https://arxiv.org/pdf/1312.... You are still MC sampling to get E_q(z|x) ln p(x | z), but I'm guessing that's easier in some sense.

1 ⌃ | ⌄ • Reply • Share ›

Yordan Hristov • a year ago

Great post, thank you!! I particularly liked the section on "*Lower bounding the Generative Model Variance*". It's something that kept me confused for quite some time since formally in all papers the covariance over x^ is defined as an output of the decoder. However, in practice we optimise the MSE between x and x^ which is still the same as optimising the likelihood p(x|z), assuming it is Gaussian, but for a **fixed** rather than **predicted** sigma.

Do you happen to have pointers to articles/papers who look into that in more depth or is it rather a practical thing that one figures out when their model does not train in a numerically stable way and they start to poke around?

Cheers.

⌃ | ⌄ • Reply • Share ›

Tridib dutta ➦ Yordan Hristov • 5 months ago

You may want to check out this paper, if you haven't already, which deals with similar issue. https://arxiv.org/pdf/2006.... . I am also interested in the **predicted** sigma. In my case, I would like to know that since I want to calculate the **probability** of the reconstructed input, specifically, what is the probability of the reconstruction input given the latent variable and that requires the sigma. The paper I have cited above, actually mentions that learning the sigma is better.

⌃ | ⌄ • Reply • Share ›

Arkadiusz Kwasigroch • 2 years ago

I wonder why the logarithm vanishes before expectation operator. Could you explain that?
\ln p(x) = \ln \int p(z) p(x \giv z) dz = \Expect_{p(z)} \brac{p(x \giv z)}

⌃ | ⌄ • Reply • Share ›

wenting zhao • 2 years ago

Your post is quite elegant too. :) Thanks a lot.

⌃ | ⌄ • Reply • Share ›

Pj • 3 years ago

Hi
Thank you for this amazing post.
I have one question though, why loss_kl is increasing, even though we are explicitly minimizing it?

⌃ | ⌄ • Reply • Share ›

Gene ➦ Pj • 2 years ago

I guess it's because we are minimising the sum of both reconstruction term and KL, and since reconstruction loss decreases much more than the amount of increase in KL, the overall loss decreases.

In other word, the model is willing to pay little increase in KL to get overall loss decreased relatively significant.

⌃ | ⌄ • Reply • Share ›

jeeyung kim • 3 years ago

Hi, Thank you for your post!
It is really great.
I have a question. I don't undestand 'Lower bounding the Generative Model Variance' part.
How can I know "our generator only returns the mean parameter but not the variance parameter"?
I think that the 'encoder' return the mean and the variance, and 'generator' return x.

• Reply • Share ›

Gene ➦ jeeyung kim • 2 years ago

In the code, there are `m, v` returned from encoder, while only `x` is returned from decoder.
If you are asking why the `x` is referred to mean parameter but not the variance of p(x|z), perhaps look at the equation

> lnp(x|z) = ...

: following the paragraph, we set variance to be one globally, so the equation is left with the the first term weighted by a constant. Since the first term describes the mean square error between x and mu(z), we know the returned x from decoder is mu(z), the mean of p(x|z).

I don't quite understand the theoretical part of setting variance of p(x|z) to one though.

• Reply • Share ›

istvan • 3 years ago • edited

Typos:
In the equation on the line after "In other words, we can build the observation model" the signma(x) should be sigma(z), doesn't it?
In the equation after "as a hypothesis that defines the joint distribution, and denote" I think it should be p_z(z), not p_z(x).

• Reply • Share ›

Rui Mod ➦ istvan • 3 years ago

Thanks; fixed!

• Reply • Share ›

istvan • 3 years ago

Why exactly is it called variational again? Simply because the variational lower bound is used and that is proved using a calculus-of-variations method? Doesn't seem to be related to [these variational methods](