Ermon Group Blog

A Tutorial on Information Maximizing Variational Autoencoders (InfoVAE)

Shengjia Zhao

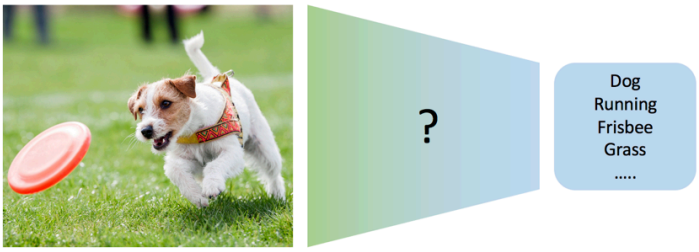
This tutorial discusses *MMD variational autoencoders* (MMD-VAE in short), a member of the InfoVAE family. It is an alternative to traditional variational autoencoders that is fast to train, stable, easy to implement, and leads to improved unsupervised feature learning.

Warm-up: Variational Autoencoding

We begin with an exposition of Variational Autoencoders (VAE in short, (Kingma and Welling 2014; Rezende, Mohamed, and Wierstra 2014)) from the perspective of unsupervised representation learning.

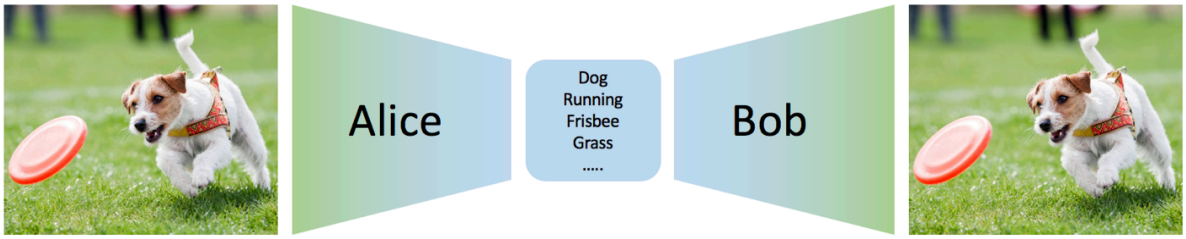
Suppose we have a collection of images, and would like to learn useful features such as object categories and other semantically meaningful attributes. However, without explicit labels or the specification of a relevant task to solve, it is unclear what constitutes

a “good feature”.



It is hard to learn good feature encoders.

One possibility is to define “good features” as features that are useful for *compressing the data*. Consider the following setting. Alice is leaving for a space exploration mission. She has a camera that can capture images with 1 million pixels. She would like to send back images to Bob (on Earth), but she can only send 100 bits of information for each image. She has to discuss a communication protocol with Bob before leaving, so that Bob can reconstruct the images as “accurately” as possible with the (short) messages he is going to receive. Luckily, they have an idea of the kind of images $x \sim p_{\text{data}}(x)$ Alice is going to see based on dataset of images collected by previous space travelers.



The autoencoding perspective of learning good feature encoders.

Let z be the 100 bit message Alice sends (which we term “latent feature”) and x be image in pixel space. To define a communication protocol, Alice needs to specify an encoding distribution $q_\phi(z|x)$ and Bob a decoding distribution $p_\theta(x|z)$. Here we consider the most general case where encoding and decoding can be randomized

procedures. Communication proceeds as follows:

1. Alice observes an image x;
2. Alice produces a distribution over messages, samples ones, and sends to Bob;
3. Bob produces a distribution over possible reconstructions based on the received message, and samples one.

In practice, encoding and decoding distributions are often modeled by deep neural networks, where ϕ and θ are the parameters of the neural net.

What remains to be formalized is what we mean by "accurate" reconstruction. This is actually an important open question (Larsen et al. 2015; Dosovitskiy and Brox 2016), but here we assume that accurate means that the original image Alice wants to send is assigned high probability $p_\theta(x|z)$ by Bob, conditioned on the received message (latent feature z).

How should we choose the encoding and decoding distributions q_ϕ and p_θ? One possibility is to jointly optimize the following reconstruction loss for each image x:

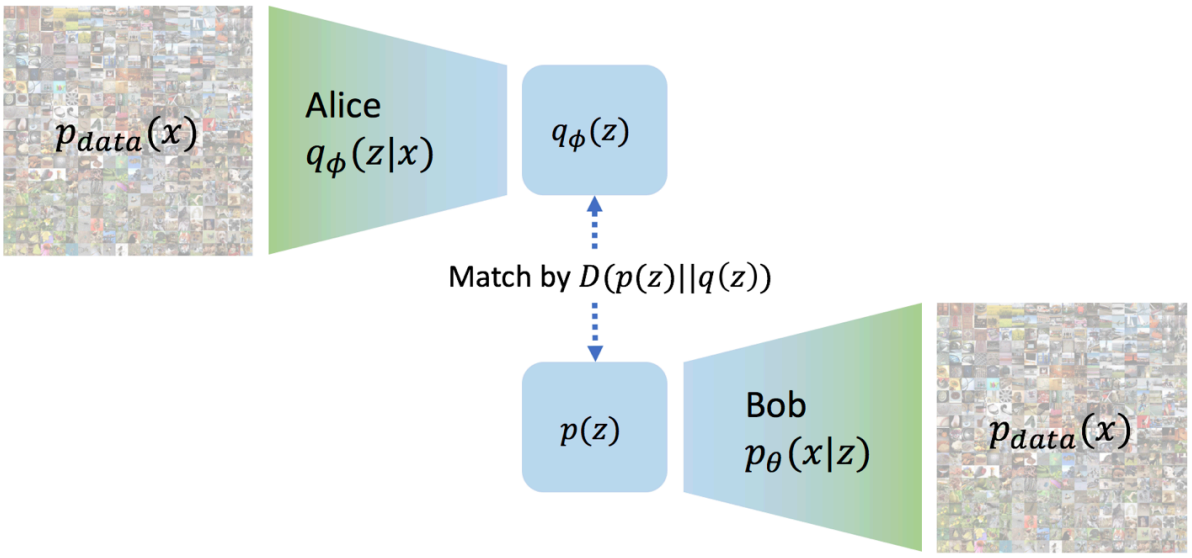
$$\mathcal{L}_{reconstruction} = \mathbb{E}_{p_{\text{data(x)}}} \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$$

This objective encourages Alice to generate good messages, so that based on the message, Bob assigns high probability to the original image. The hope is that if Alice and Bob can accomplish this, the message (latent feature) should contain the most salient features and capture the main factors of variation in the data.

In addition to this, we may want to enforce additional structure on

the message space. For example, Alice should be able to observe an image, generate latent features, change only a part of it (such as certain attributes), and Bob should still be able to generate sensible outputs reflecting the change. It would also be nice for Alice to be able to directly generate valid messages without having to observe actual images.

To do this we must define the space of valid messages. This can be achieved by defining a "prior" distribution of valid messages $p(z)$. $p(z)$ is usually a very simple distribution such as a uniform or Gaussian distribution. Accounting for all possible images from the true underlying distribution $p_{data}(x)$, we get a distribution over possible messages Alice generates $q_\phi(z) = \mathbb{E}_{p_{data}(x)}[q_\phi(z|x)]$. We would like the message distribution Alice generates to match the distribution on valid messages $p(z)$.



Matching distributions of messages.

We thus get a general family of variational auto-encoding objectives

$$\mathcal{L}_{VAE} = -\lambda D(q_\phi(z)\|p(z)) + \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$$

where D is any strict divergence, meaning that $D(q\|p) \geq 0$ and

$D(q\|p) = 0$ if and only if $q = p$, and $\lambda > 0$ is a scaling coefficient.

Which divergence should one choose? The original VAE objective (Kingma and Welling 2014) uses $\mathbb{E}_{p_{data}(x)}[-\mathrm{KL}(q_\phi(z|x)\|p(z))]$, which is minimized if the message $q_\phi(z|x)$ Alice generates for **each** input x matches the prior $p(z)$. Of course, this implies that the distributions also match when "averaging" over all possible inputs. This can be problematic in some scenarios. For example, it is not clear why we would want Alice to generate the same message for each possible input, as this works against our goal of learning good features. We will return to this crucial aspect later. For now, we introduce an alternative divergence called Maximum Mean Discrepancy (Gretton et al. 2007), which we will show has many advantages compared to traditional approaches.

Maximum Mean Discrepancy

Maximum mean discrepancy (MMD, (Gretton et al. 2007)) is based on the idea that two distributions are identical if and only if all their moments are the same. Therefore, we can define a divergence by measuring how "different" the moments of two distributions $p(z)$ and $q(z)$ are. MMD can accomplish this efficiently via the kernel embedding trick:

$$\mathrm{MMD}(p(z)\|q(z)) = \mathbb{E}_{p(z),p(z')}[k(z,z')] + \mathbb{E}_{q(z),q(z')}[k(z,z')] - 2\mathbb{E}_{p(z),q(z')}[k(z,z')]$$

where $k(z,z')$ is any universal kernel, such as Gaussian $k(z,z') = e^{-\frac{\|z-z'\|^2}{2\sigma^2}}$. A kernel can be intuitively interpreted as a function that measures the "similarity" of two samples. It has a large value when two samples are similar, and small when they are different. For example, the Gaussian kernel considers points that are close in Euclidean space to be "similar". A rough intuition of MMD, then, is that if two distributions are identical, then the average

"similarity" between samples from each distribution, should be identical to the average "similarity" between mixed samples from both distributions.

Why Use an MMD Variational Autoencoder

In this section we argue in detail why we might prefer MMD-VAE over the traditional evidence lower bound (ELBO) criterion used in VAEs:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{p_{data}(x)}[-\text{KL}(q_\phi(z|x)||p(z))] + \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$$

We show that ELBO suffers from two problems that MMD-VAE

$$\mathcal{L}_{\text{MMD-VAE}} = \text{MMD}(q_\phi(z)||p(z)) + \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$$

does not suffer from.

Problem 1: Uninformative Latent Code

Researchers have noticed that the term

$$\mathbb{E}_{p_{data}(x)}[-\text{KL}(q_\phi(z|x)||p(z))]$$

might be too restrictive (Chen et al. 2016; Bowman et al. 2015; Sønderby et al. 2016). Intuitively it encourages the message $q_\phi(z|x)$ to be a random sample from $p(z)$ for each x, making the message uninformative about the input. If the decoder is very flexible, then a trivial strategy can globally maximize ELBO: Alice only produces $p(z)$ regardless of the input, and Bob only produces $p_{data}(x)$ regardless of Alice's message. This means that we have failed to learn any meaningful latent representation. Even when Bob cannot generate a complex distribution (e.g. $p_\theta(x|z)$ is a Gaussian), this still

encourages the model to under-utilize the latent code. Several methods have been proposed (Chen et al. 2016; Bowman et al. 2015; Sønderby et al. 2016) to alleviate this problem, but they involve additional overhead, and do not completely solve the issue.

MMD Variational Autoencoders do not suffer from this problem. As shown in our paper, it **always prefers to maximize mutual information between x and z**. Intuitively, the reason is that the distribution over messages has to match the prior only in expectation, rather than for every input.

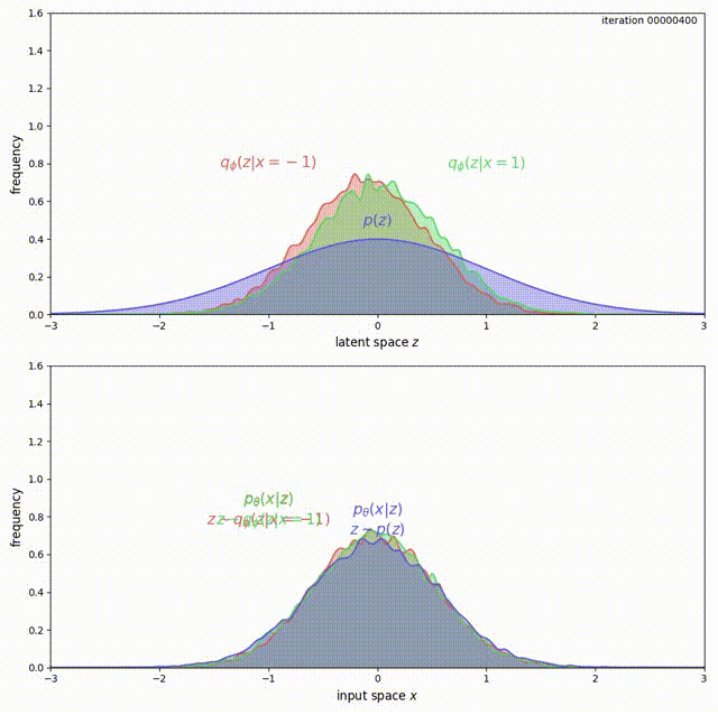
Problem 2: Variance Over-estimation in Feature Space

Another problem with ELBO-VAE is that it tends to over-fit data, and as a result of the over-fitting, learn a $q_\phi(z)$ that has variance tending to infinity. As a simple example, consider training ELBO on a dataset with two data points $\{-1, 1\}$, and both encoder $q_\phi(z|x)$ and decoder $p_\theta(x|z)$ output Gaussian distributions with non-zero variance. We prove in our paper that $\mathcal{L}_{ELBO}$ can be maximized to infinity by

1. The mean of $q_\phi(z|x=-1)$ tends to $-\infty$, and the mean of $q_\phi(z|x=1)$ tends to $+\infty$.
2. The probability density of $p_\theta(x|z<0)$ is concentrated around $x=-1$, and $p_\theta(x|z>0)$ is concentrated around $x=1$.

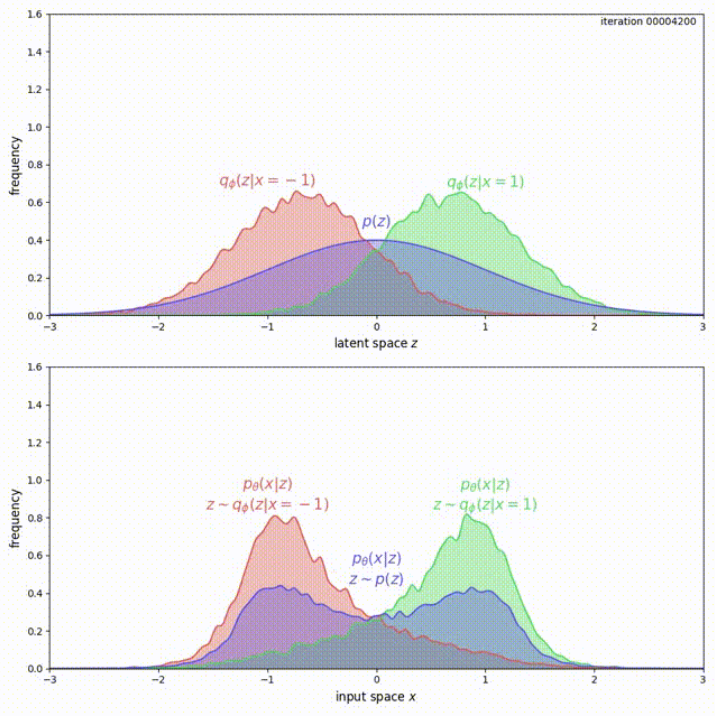
The problem here is that, for ELBO, the regularization term $\mathrm{KL}(q_\phi(z|x)||p(z))$ is not strong enough compared to the reconstruction loss. Instead of forcing $q_\phi(z) = \frac{1}{2}q_\phi(z|x=-1) + \frac{1}{2}q_\phi(z|x=1)$ to match $p(z)$, it prefers to have two modes that are pushed infinitely far from each other. Intuitively, pushing the modes as far as possible from each other reduces ambiguity during reconstruction (the messages

corresponding to the two inputs are clearly separated, hence Bob's decoding is easy). Although there is a preference to match the prior, the reconstruction term in the ELBO objective dominates. This can be experimentally verified and visualized: observe how mass of $q_\phi(z|x)$ is pushed away from 0 (top animation) as the model overfits this small dataset (bottom animation).



Mass is pushed away in the case of ELBO.

We cannot simply add a large coefficient β to $\mathrm{KL}(q_\phi(z|x)||p(z))$. Even though adding a coefficient β larger than 1 has been shown to improve generalization (Higgins et al. 2016), this is exactly the term that encourages the model not to use the latent code (Problem 1). Increasing it will make matters worse for the uninformative latent code problem. On the other hand, with the MMD-VAE objective we can freely increase the weight of the regularization term $\lambda D(q_\phi(z)||p(z))$, with little negative consequences. As shown below, we get much better behavior when $\lambda = 500$ in the previous example:



MMD has much better behavior.

This matters in practice when the dataset is small. For example, when training on MNIST with only 500 examples, ELBO overfits and generates poor samples *(Top)*, while InfoVAE generates reasonable (albeit blurry) samples *(Bottom)*.

Samples from original VAE.

Samples from InfoVAE.

Implementing a MMD Variational Autoencoder

The code for this tutorial can be downloaded here, with both python and ipython versions available. The code is fairly simple, and we will only explain the main parts below. (A pytorch version provided by Shubhanshu Mishra is also available.)

To efficiently compute the MMD statistics and exploit GPU parallelism, we use the following code

```
def compute_kernel(x, y):
    x_size = tf.shape(x)[0]
    y_size = tf.shape(y)[0]
    dim = tf.shape(x)[1]
    tiled_x = tf.tile(tf.reshape(x, tf.stack([x_size, 1, dim])), tf.stack
    tiled_y = tf.tile(tf.reshape(y, tf.stack([1, y_size, dim])), tf.stack
    return tf.exp(-tf.reduce_mean(tf.square(tiled_x - tiled_y), axis=2) /

def compute_mmd(x, y, sigma_sqr=1.0):
    x_kernel = compute_kernel(x, x)
    y_kernel = compute_kernel(y, y)
    xy_kernel = compute_kernel(x, y)
```

```
    return tf.reduce_mean(x_kernel) + tf.reduce_mean(y_kernel) - 2 * tf.r
```

The first function `compute_kernel(x, y)` takes as input two matrices `(x_size, dim)` and `(y_size, dim)`, and returns a matrix `(x_size, y_size)` where the element (i, j) is the outcome of applying the kernel to the i-th vector of x, and j-th vector of y. Given this matrix, we can compute the MMD statistics according to the definition. `sigma_sqr` controls the smoothness of the kernel function, which is a hyper-parameter σ^2. We find MMD to be fairly robust to its selection, and that using $2/dim$ is a good option in our experiments.

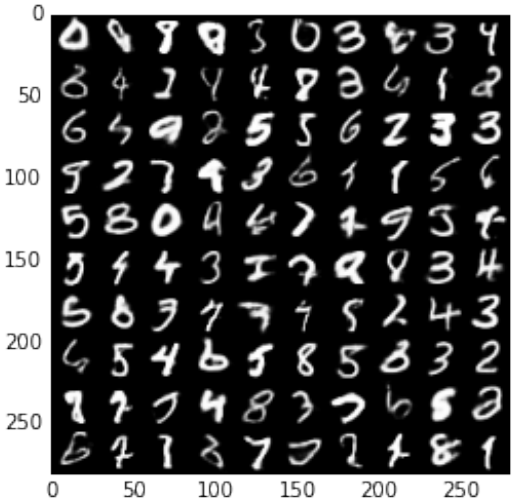
To match samples from the prior $p(z)$ and from the encoding distribution, we can simply generate samples from the prior distribution $p(z)$, and compare the MMD distance between the real samples and the generated latent codes.

```
true_samples = tf.random_normal(tf.stack([200, z_dim]))
loss_mmd = compute_mmd(true_samples, train_z)
```

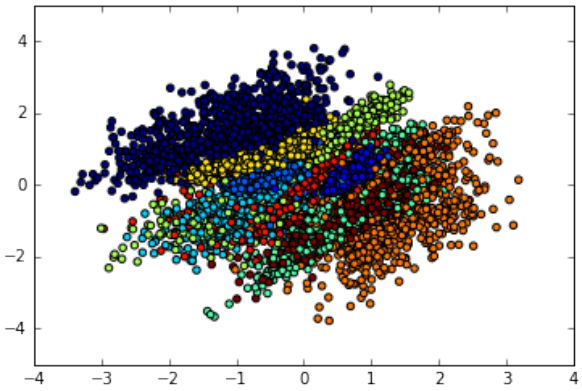
We suppose that the distribution Bob produces $p_\theta(x|z)$ is a factorized Gaussian. The mean of the Gaussian $\mu(z)$ is produced by a neural network, and the variance is simply assumed to be some fixed constant. The negative likelihood is thus proportional to the squared distance $\|x - \mu(z)\|^2$. The total loss is a sum of this negative log likelihood and the MMD distance.

```
loss_nll = tf.reduce_mean(tf.square(train_xr - train_x))
loss = loss_nll + loss_mmd
```

Training on a Titan X for approximately one minute already gives very sensible samples.



Furthermore if the latent code only has two dimensions, we can visualize the latent code produced by different digit labels. We can observe good disentangling.



Each color is a digit class. It seems that the learned features are disentangled.

References

1. Kingma, Diederik P, and Max Welling. 2014. “Auto-Encoding Variational Bayes.” In *International Conference on Learning Representations*.
2. Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra. 2014. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models.” *ArXiv Preprint ArXiv:1401.4082*.