University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 20: November 1

Lecturer: Alexander Schwing Scribe: Yu Tao

1 Overview

This lecture introduces Variational Auto-Encoders (VAE). A Variational Auto-Encoders resembles a classical auto-encoder and is a neural network that consists of an encoder, a decoder and a loss function. The difference is that a variational autoencoderprovides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, we'll formulate our encoder to describe a probability distribution for each latent attribute [4]. It can be used for image generation. For example, a practical application may be to generate trees for a forest in a video game, which are all similar but not the same [1].

The goal of this lecture includes:

- Getting to know Variational Auto-Encoders
- Understanding generative methods
- Differentiating between discriminative and generative methods

2 Introduction

2.1 Latent Variables

When training a generative model, things become difficult as the dependencies between the dimensional become more complicated. Take, handwritten characters generating (0-9), for example, it helps if the model first decides which character to generate before it assigns a value to any specific pixel. This kind of decision is formally called a *latent variable*. So before we draw any character, we will first randomly samples a digit value z from the set [0,...,9], and then makes sure all the strokes match that character [3].

In other words, the latent space is the space where your features lie.

2.2 Auto-Encoder

Neural networks exist in all shapes and sizes, and are often characterized by their input and output data type. For instance, image classifiers are built with Convolutional Neural Networks. They take images as inputs, and output a probability distribution of the classes.

Autoencoders (AE) are a family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.

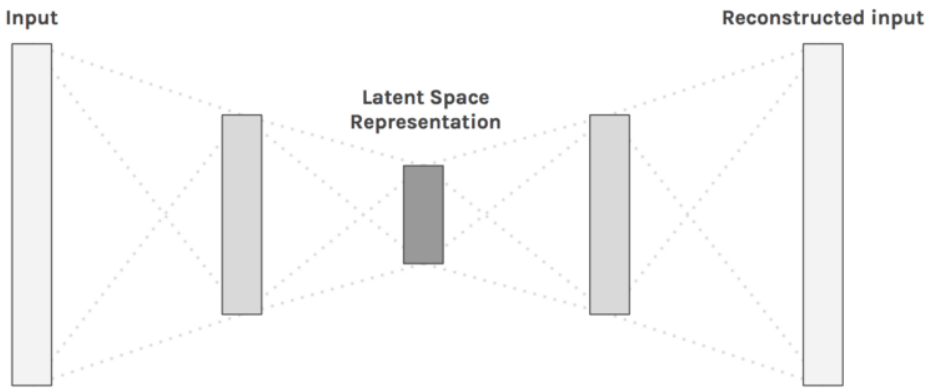


Figure 1: Simple Autoencoder Architecture -The input is compressed and then reconstructed [2].

2.3 Variational Auto-Encoders

Let's suppose we've trained an autoencoder model on a large dataset of faces with a encoding dimension of 6. An ideal autoencoder will learn descriptive attributes of faces such as skin color, whether or not the person is wearing glasses, etc. in an attempt to describe an observation in some compressed representation.

However, we may prefer to represent each latent attribute as a range of possible values. Using a variational autoencoder, we can describe latent attributes in probabilistic terms.

With this approach, we'll now represent each latent attribute for a given input as a probability distribution. When decoding from the latent state, we'll randomly sample from each latent state distribution to generate a vector as input for our decoder model. [4].

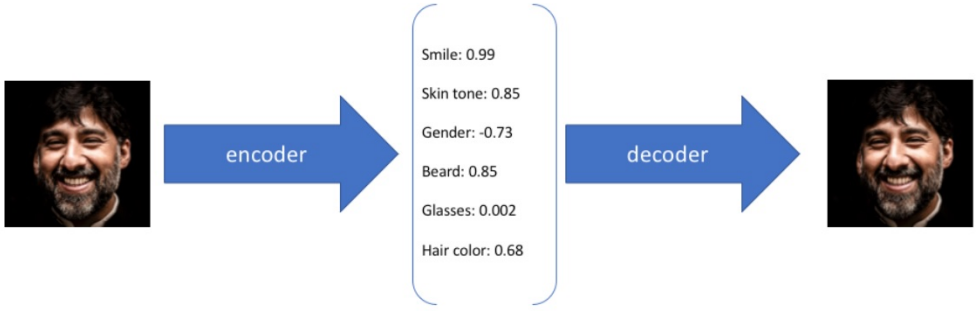


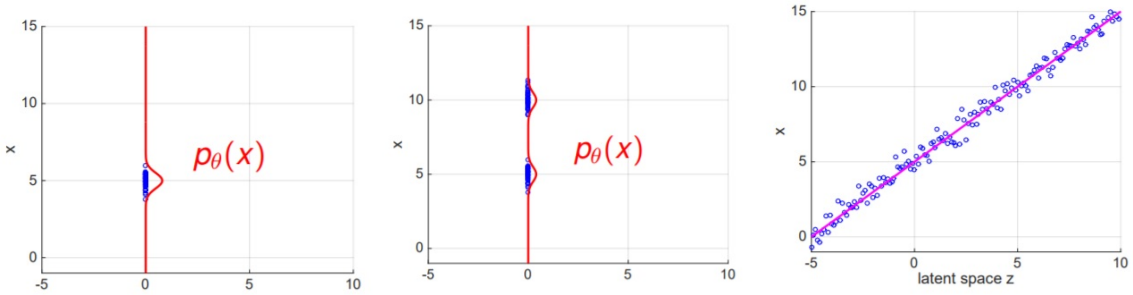
Figure 2: Latent Attributes

3 Construct a Probability Distribution

3.1 Modeling a Data Distribution $p(x)$

Then how about modeling a distribution $p(x)$ for the data? Given data points, we can model $p(x)$ by

- Fit mean and variance (parameter θ) of a distribution (e.g., Gaussian), see Figure 3(a)



(a) Example 1 (b) Example 2 (c) Example 3

Figure 3: Example of $p(x)$

- Fit parameters θ of a mixture distribution (e.g., mixture of Gaussian, k-means), see Figure 3(b)

But sometimes we don't know how many components there are (see Figure 3(c)). In this case, we can assume our data to originate from some more low dimensional spaces. If someone gives us latent space z, we can model $p(x|z)$ instead of modeling $p(x)$.

$$p_\theta(x|z) \propto exp(-\frac{x-\mu(x)^2}{2\sigma^2}) \tag{1}$$

3.2 Low-dimensional Manifold Hypothesis

Manifold Hypothesis:

- x is a high dimensional vector
- our data arises from low dimensional subspace, which is z

Here is an example of low-dimensional manifolds. Imagine your z was confined to the interpolation between 0 and 1. Now you are mapping this value z to the 2D space. All the blue data points are concentrated on the line and now we need to figure out how can we find the z space and how can we map into this high dimensional space and the we can describe all those points. We can fit many Gaussians to it but the computation can be very expensive. So the main idea of this manifolds assumption was that given a value z, then it would be very easy to reconstruct.

One object we already saw is $p_\theta(x|z)$. we need to figure out the distribution that allows us to recover x. Let's have a z from a sample Gaussian and transformed by deep net to recover x, which is called decoder, see Figure 5.

4 Training

Auto-encoder takes your data x, compresses it into z space, and reconstructs your data from z.

Now we need to know how to get from x to z, which is $p_\theta(z|x)$. Since we know $p_\theta(x|z)$ already, we can write it in this way, using Bayes' theorem.

$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)} \tag{2}$$

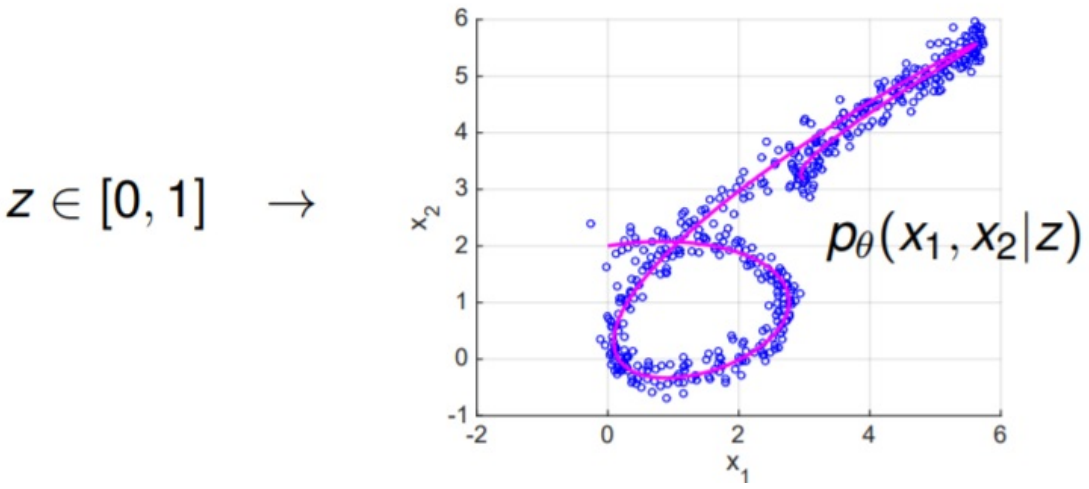


Figure 4: Example of low-dimensional manifolds

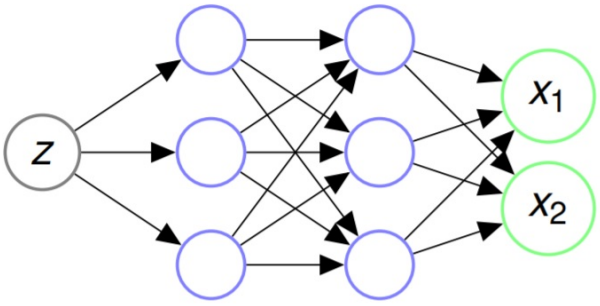


Figure 5: Decoder Architecture

Given $p_\theta(x|z)$ we can get

$$p_\theta(x) = \int_{\hat{z}} p_\theta(x|\hat{z})p(\hat{z})d\hat{z} \tag{3}$$

Combine equation (3) and (4) we now have

$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{\int_{\hat{z}} p_\theta(x|\hat{z})p(\hat{z})d\hat{z}} \tag{4}$$

The issue here is we cannot write down a close form solution for the integral. So how do we deal with it? It's exactly the same thing we are doing in machine learning. if we cannot not solve it, we approximate it. There are two approaches to approximate $p_\theta(z|x)$:

- Sampling techniques (draw a bunch of samples, compute that integral by summing up all your points, which is very costly)
- Approximate $p_\theta(z|x)$ with $q_\phi(z|x)$ using another deep network

Since the distribution z has to be a very simple distribution of a Gaussian, the output of the q is also just a Gaussian. Given x, we want to get mean and variance.

$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x)) \tag{5}$$

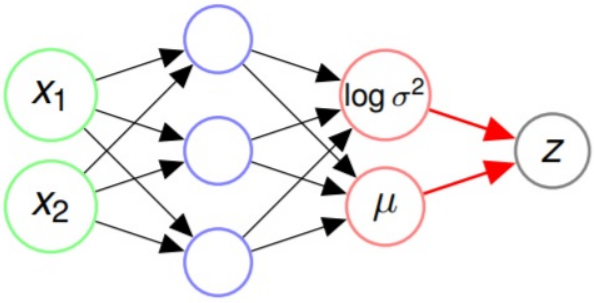


Figure 6: Encoder Architecture

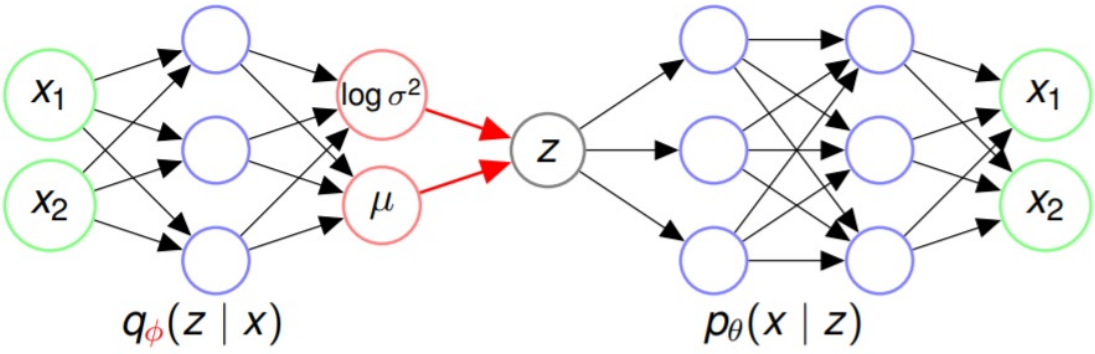


Figure 7: Variational Auto-encoder Architecture

As we can see from Figure 6, x goes in as input, which is our training data. We get mean and log sigma squared, from which we draw sample z. And we feed sample z in our decoder model, then we get our x as output.

Since we get z by sampling, we cannot back-propagate through it because sampling is not differentiable. But we can use Reparameterization trick to solve this issue:

$$z \sim q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x)) \tag{6}$$

which is equivalent to:

$$z = \mu_\phi(x) + \sigma_\phi(x) \cdot \epsilon \tag{7}$$

where $\epsilon \sim \mathcal{N}(0, 1)$

Now our architecture looks like this (Figure 8):

We now draw samples from ϵ but that doesn't depend on any parameters of any of your dependencies. So you can compute the derivative with respect to μ and $log\sigma^2$.

5 Optimize Loss Function

Given data x, we want to maxmize the likelihood. How do we model $logp_\theta(x)$?

We introduce z and marginalize it.

$$logp_\theta(x) = \int_z q_\phi(z|x) logp_\theta(x) \tag{8}$$

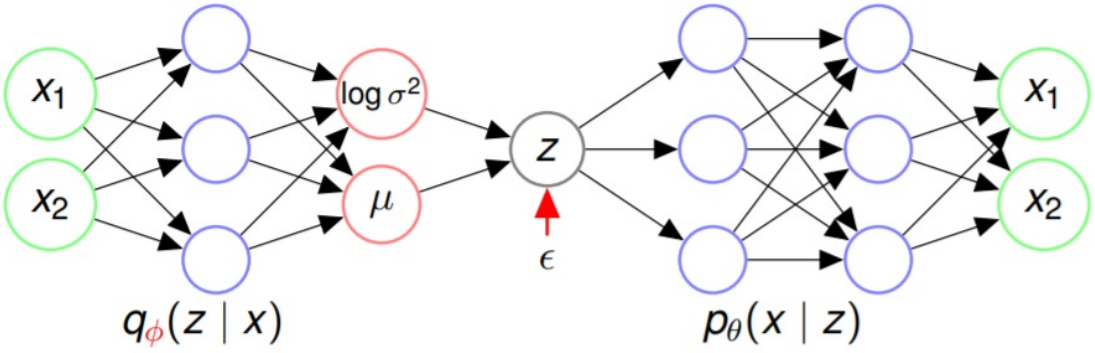


Figure 8: Variational Auto-encoder Architecture with Epsilon

Replace it with joint distribution we have

$$logp_\theta(x) = \int_z q_\phi(z|x) log \frac{p_\theta(x,z)}{p_\theta(z|x)} \tag{9}$$

We introduce $q_\phi(z|x)$ then the equation becomes

$$logp_\theta(x) = \int_z q_\phi(z|x) log (\frac{p_\theta(x,z)}{q_\phi(z|x)} \cdot \frac{q_\phi(z|x)}{p_\theta(z|x)}) \tag{10}$$

Then we split it into two parts

$$\begin{aligned} logp_\theta(x) &= \int_z q_\phi(z|x) log \frac{p_\theta(x,z)}{q_\phi(z|x)} + \int_z q_\phi(z|x) log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\ &= \mathcal{L}(p_\theta, q_\phi) + D_{KL}(q_\phi, p_\theta) \end{aligned} \tag{11}$$

$D_{KL}(q_\phi, p_\theta)$ is non-negative so we have lower bound and we want ot maximize the lower bound.

$$logp_\theta(x) \geq \mathcal{L}(p_\theta, q_\phi) \tag{12}$$

We know $p_\theta(x|z)$ already and we can also get $q_\phi(z|x)$ by feeding data x into our encoder and compute mean and variance. But how do we know the joint distribution $p_\theta(x,z)$.

5.1 Approximate Inference

$$\mathcal{L}(p_\theta, q_\phi) = \int_z q_\phi(z|x) log \frac{p_\theta(x,z)}{q_\phi(z|x)} \tag{13}$$

Just like what we did before, we replace $p_\theta(x,z)$ with $p_\theta(x|z)p(z)$. Then we have

$$\mathcal{L}(p_\theta, q_\phi) = \int_z q_\phi(z|x) log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} \tag{14}$$

It's more convenient to write it into separated parts, so we get

$$\mathcal{L}(p_\theta, q_\phi) = \int_z q_\phi(z|x) log \frac{p(z)}{q_\phi(z|x)} + \int_z q_\phi(z|x) log p_\theta(x|z) \tag{15}$$

The first part is

$$\int_z q_\phi(z|x) log \frac{p(z)}{q_\phi(z|x)} = -D_{KL}(q_\theta, p) \tag{16}$$

And the second part is

$$\int_z q_\phi(z|x) log p_\theta(x|z) = \mathbb{E}_{q_\phi}[log p_\theta(x|z)] \tag{17}$$

With equation 15, 16 and 17, we have

$$\mathcal{L}(p_\theta, q_\phi) = -D_{KL}(q_\theta, p) + \mathbb{E}_{q_\phi}[log p_\theta(x|z)] \tag{18}$$

and this is exactly what we want to maximize.

One thing to point out here is that we cannot throw away the KL Divergence here because we are optimizing with respect to parameter ϕ. $D_{KL}(q_\theta, p)$ is non-negative and $D_{KL}(q_\theta, p)$ is negative, so we would not get the lower bound if we drop it (you will increase the value).

$D_{KL}(q_\theta, p)$ is a regularization term with prior $p(z)$, which is often a Gaussian. $\mathbb{E}_{q_\phi}[log p_\theta(x|z)]$ is the reconstruction part. You will feed in a z, and then you want to reconstruct data x.

5.2 Regularization

$$-D_{KL}(q_\theta, p) = \int_z q_\phi(z|x) log \frac{p(z)}{q_\phi(z|x)} \tag{19}$$

- $p(z) = \mathcal{N}(z; 0, 1)$
- $q\phi(z|x)$ is Gaussian with mean $\mu_\phi(x)$, variance $\sigma^2_\phi(x)$

5.3 Reconstruction

$$\mathbb{E}_{q_\phi}[log p_\theta(x|z)] = \int_z q_\phi(z|x) log p_\theta(x|z) \tag{20}$$

we can approximate $\mathbb{E}_{q_\phi}[log p_\theta(x|z)]$ by drawing samples from distribution $q_\phi(z|x)$, computing the log and averaging them like this

$$\mathbb{E}_{q_\phi}[log p_\theta(x|z)] \approx \frac{1}{N} \sum_{i=1}^{N} log p_\theta(x|z^i) \tag{21}$$

where $z^i \sim \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x))$, z is from q and is normal distribution.

Now we know how to optimize the loss function

$$\mathcal{L}(p_\theta, q_\phi) \approx -D_{KL}(q_\theta, p) + \frac{1}{N} \sum_{i=1}^{N} log p_\theta(x|z^i) \tag{22}$$

and we can compute the gradients with respect to any of its parameters for the first part and we can compute the gradients for the second part by back-propagating through whatever deep nets we use.

Intuitively, we get our data point x, and we feed that into our encoder. Our encoder gives us some distribution over z, which is q_ϕ. From that distribution q we draw some samples and we feed the samples into the decoder to check how likely it is with our original data.

Note that in many implementations, typically N is very small, for example 1. If you want a better approximation, draw more samples, but obviously we will need more computation and memory on this.

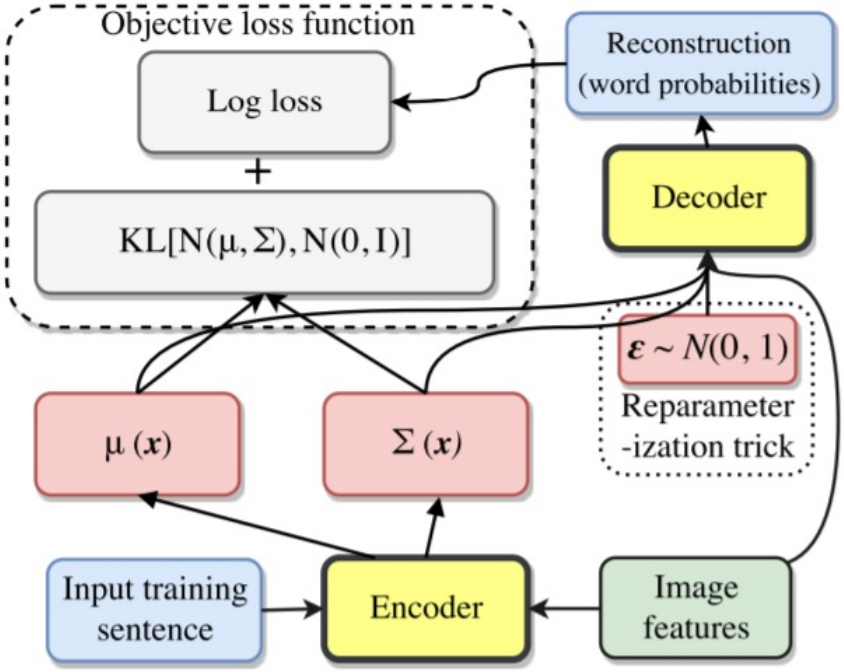


Figure 9: VAE flowchart

References

[1] Variational autoencoders. https://www.doc.ic.ac.uk/ js4416/163/website/autoencoders/variational.html.

[2] J. Despois. Autoencoders. https://ai-odyssey.com/2017/02/07/autoencoders

[3] C. Doersch. Tutorial on variational autoencoders. abs/1606.05908, 2016.

[4] J. Jordan. Variational autoencoders. https://www.jeremyjordan.me/variational-autoencoders/, 2018.