University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 20: November 14

Lecturer: Alexander Schwing Scribe: Siddhartha Satpathi

1 Introduction

This lecture is about modelling an unknown distribution $p(x)$ from the data $X_1, \ldots, X_n$. We want to learn the unknown generating distribution $p(x)$ from the data so that we can generate new data points X *like* the existing data points $X_1, \ldots, X_n$ but not exactly the same. One naive way to get $p(x)$ would be to provide high probability to points in the set $X_1, \ldots, X_n$ and zero everywhere else, but this would not allow us to generate new points similar to, but not exactly equal to the old data points. As as example, we can generate new pictures of faces from an existing image of faces.

Let us discuss some very simple ways to model $p(x)$. Suppose the data points $X_1, \ldots, X_n$ look like the ones in Figure 2. Suppose we try and use Gaussian distribution with an unknown mean and variance to model the data points. By, method of moments, or maximum likelihood, we can estimate the mean and variance to be the sample mean and sample variance of the data points $X_1, \ldots, X_n$. $\theta = (\mu, \sigma)$ is the parameter that we learn from the data points for the class of normal distributions. Figure 2 shows one possible fitting of Gaussian distribution to the right.

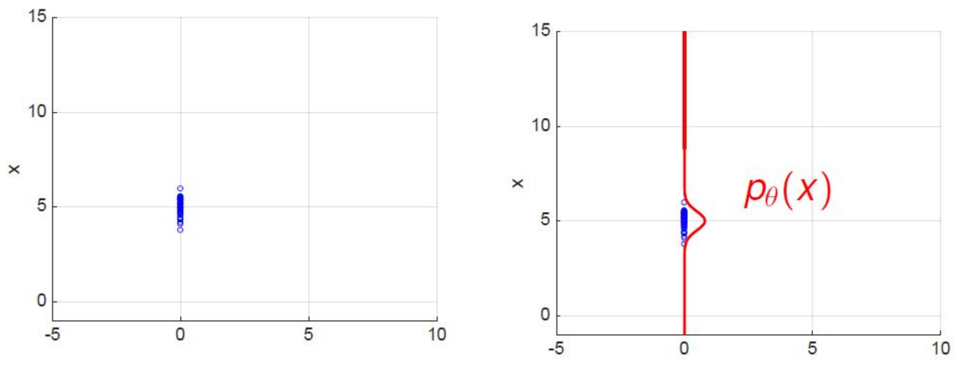


Figure 1: Data points $X_1, \ldots, X_n$ from a Gaussian distribution

Similarly, suppose the data comes from a mixture of two Gaussian distributions as shown in the left of Figure **??**. Suppose the parameter θ represents the two means and the probability of which Gaussian distribution is the sample from. So $\theta = \mu_1, \mu_2, \sigma_1, \sigma_2, \alpha$. We can do a maximum likelihood for the parameters, but that would be hard to compute α. So we can go for an iterative scheme, the Expectation Maximization approach to estimate θ.

Now suppose random variable X, Z have a joint distribution $p_\theta(x, z)$. But we only see data samples $X_1, \ldots, X_n$ from the marginal $p_\theta(x)$. Here the conditional distribution of X given Z, $p_\theta(x|z)$, is $\mathcal{N}(z, \sigma^2)$ and the distribution of marginal $p(z)$ is $Uniform(-5, 10)$. If we just observe samples $X_1, \ldots, X_n$ from the marginal $p_\theta(x)$ as shown in Figure 3 (left) it seems very difficult to fit a parametric class of distributions. But upon introducing a hidden random variable z (Figure 3 middle) we can estimate the parameter set σ^2 and explain the data points. If we have to generate a new data point X we first sample z from $p(z)$ and then sample from $p_\theta(x|z)$. In general X can be from a high dimensional space but $p_\theta(x|z)$ has high probability for only a small set inside $\mathcal{Z}$ (Say $\mathcal{Z}$ is the support space of the distribution Z). For example, suppose X lies in $\mathbb{R}^2$ and Z is in $\mathbb{R}$.

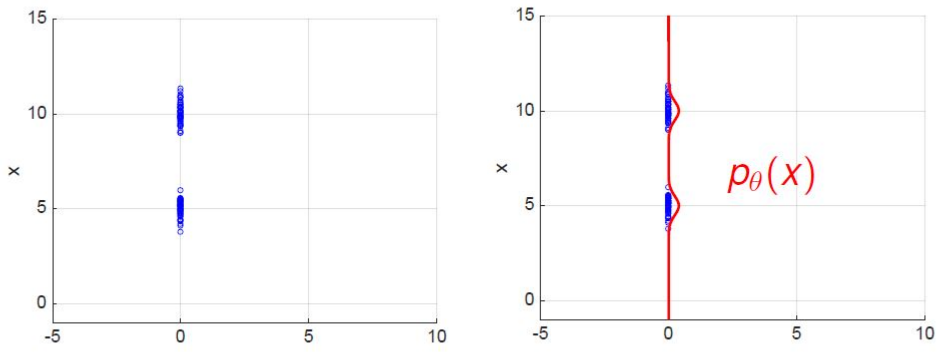


Figure 2: Data points $X_1, \dots, X_n$ from mixture of Gaussians

But $p_\theta(x|z)$ is only positive for $z \in [0, 1]$. This example conditional distribution is shown in Figure 4 below, Auto encoders are way to generating these conditional distributions $p_\theta(x|z)$ where the θ lies in the space of neural network architecture with non linear neurons. We motivate Variational Autoencoders in the next section.

2 Notation

The KL divergence between two continuous distributions p and q is given by

$$\mathcal{D}_{KL}(p||q) = \int q(x) \log \frac{q(x)}{p(x)} = \mathbb{E} \log \frac{q(x)}{p(x)}.$$

The KL divergence is always positive and it is equal to 0 only when the distributions p and q are identical. Note that the KL divergence is not symmetric and hence it is not a distance. The trace of a matrix A is the sum of diagonal elements of the matrix and is denoted by $\text{Tr}(A)$.

3 Motivation for Variational Autoencoders (VAE)

Suppose we have n i.i.d. samples $X_i, i = 1, \dots, n$ from the distribution of X. The distribution of X is given by $p_\theta(x)$, which is parameterized by θ. θ is assumed to lie in the space Θ. An example of a distribution family $\{p_\theta(x), \theta \in \Theta\}$ is a Gaussian family $\mathcal{N}(\theta, 1)$. This example is very restrictive to model complex data. So in general if we use a Θ as a neural network with lots of neurons, we can hopefully fit any complex distribution on a large data set $X_i, i = 1 \dots n$. Suppose the data is generated through an unobserved continuous random variable $z \in \mathcal{Z}$ from distribution $p_\theta(z)$. Our goal is to find the distribution which maximizes the likelihood of the data $X_i, i = 1 \dots n$. Ideally, we want to compute:

$$\max_\theta \log p_\theta(X) = \int p_\theta(x|z) p_\theta(z) dz. \tag{1}$$

1. Equation 1 can be intractable because we might not be able to get a closed form solution to the integration or we would not be able to differentiate it w.r.t θ. Hence we cannot perform the *maximum likelihood* approach.

2. If we attempt to use the EM algorithm from Lecture 18 to find an iteratve solution to the maximum likelihood problem, we need to be able to compute the posterior density $p_\theta(z|x)$

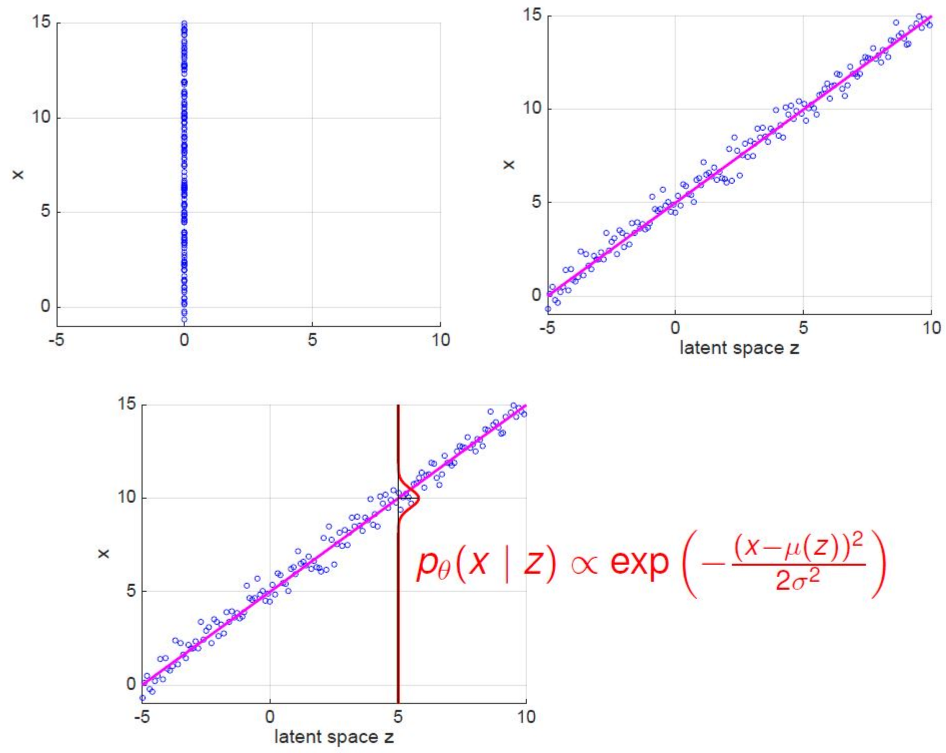


Figure 3: Data points $X_1, \ldots, X_n$ with hidden random variable z

for the E- step of the algorithm. But

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

which can be intractable.

3. When n is large we might not be able to approximate the distribution $p_\theta(z|x)$ from samples through Markov Chain Monte Carlo (MCMC) techniques, because obtaining many samples can be costly as the mixing time of the Markov chain might be high or the Markov chain gets stuck producing samples from some part of set $\mathcal{Z}$.

Since $p_\theta(z|x)$ is unknown, in VAE we try to approximate $p_\theta(z|x)$ with $q_\phi(z|x)$ (the encoder).

3.1 Idea behind Variational Autoencoder (VAE)

In general the space of $\mathcal{Z}$ can be large and for most values of $z \in \mathcal{Z}$ the conditional distribution $p_\theta(x|z)$ might be 0. The key idea behind VAE is to attempt to sample z from the small set of region which gives high probability to $p_\theta(x|z)$. The approximation $q_\phi(z|x)$ to $p_\theta(z|x)$ is used to generate these good samples of z (the encoder part) and then the $p_\theta(x|z)$(the decoder) is used to get samples from X. See Figure 5.

In general $q_\phi(z|x)$ and $p_\theta(x|z)$ can both be deep neural networks. Let us take a simple example to demonstrate this.

Suppose z is a one dimensional Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. We approximate the function $p_\theta(x|z)$ of x with input z as the a deep neural network (this is the decoder). In general, if the

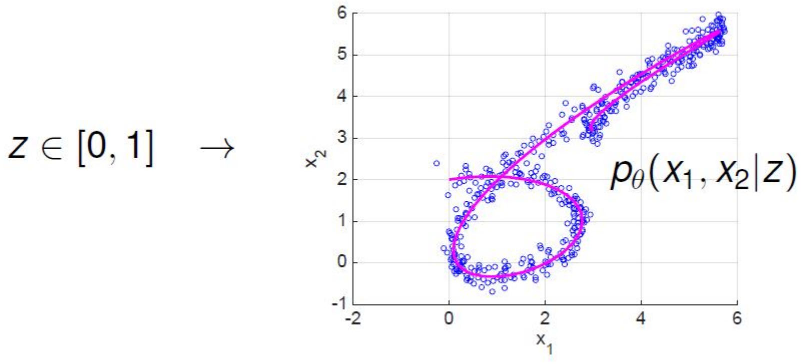


Figure 4: Low dimensional hidden variable space $\mathcal{Z}$

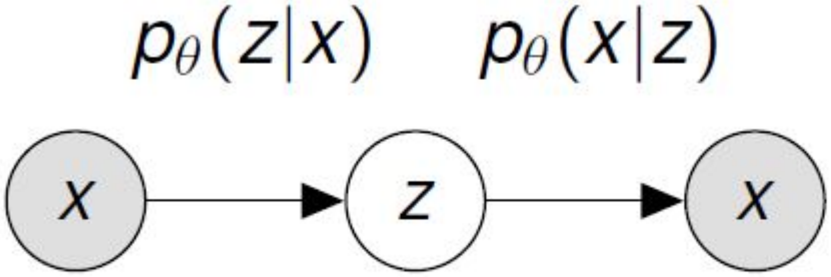


Figure 5: A Variational Autoencoder

deep neural network can approximate very complex functions, we can get it match the distribution function $p_\theta(x|z)$ with good accuracy. Say the neural network is given in the Figure 6 below. Now, if magically we are able to get the neural network to learn the function $p_\theta(x|z)$, we get a way to generate samples from x by inputting the decoder samples from z. Note that, we do not actually know the distribution of z, because of the parameters μ, σ are unknown to us. We only have access to data points $X_1, \ldots, X_n$. We would like to compute μ, σ from the data points $X_1, \ldots, X_n$. Say we have another neural network which approximates μ, σ from data points $X_1, \ldots, X_n$ as shown in Figure **??**. This would be the encoder.

In order to generate new sample X like the ones $X_1, \ldots, X_n$, we can feed encoder function to an arbitrary X. The output z can in turn be fed to the decoder in Figure 6 to generate samples from random variable X.

4 Loss function and Variational Lower bound

Since maximizing the likelihood function $p_\theta(x)$ might be intractable, we approximate it by maximizing a lower bound to $p_\theta(x)$. Lemma 1 below derives the empirical lower bound (ELBO) $\mathcal{L}(p_\theta, q_\phi)$ to $\log p_\theta(x)$.

Lemma 1 *We have the indentity*

$$\log p_\theta(x) - \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x)) = \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) - \mathcal{D}_{KL}(q_\phi(z|x)||p(z)). \tag{2}$$

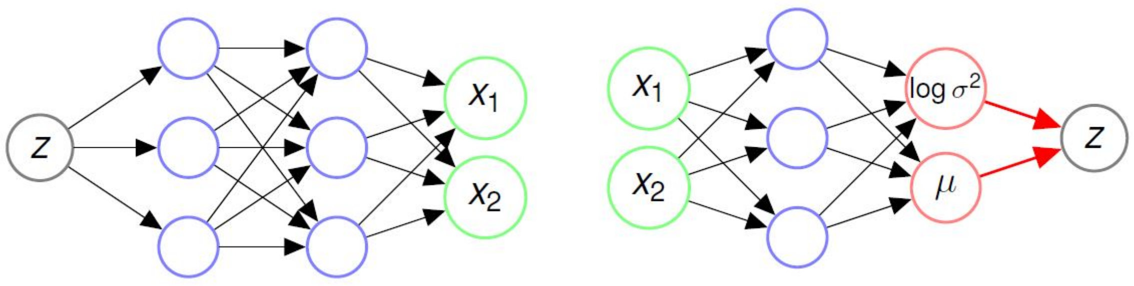


Figure 6: An example decoder(left) and encoder(right)

Proof: We have that,

$$\log p_\theta(x) = \log p_\theta(x) \times \int_z q_\phi(z|x) dz \tag{3}$$
$$= \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{p_\theta(z|x)} dz \tag{4}$$
$$= \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} \frac{q_\phi(z|x)}{p_\theta(z|x)} dz \tag{5}$$
$$= \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} dz + \int_z q_\phi(z|x) \frac{q_\phi(z|x)}{p_\theta(z|x)} dz \tag{6}$$
$$= \mathcal{L}(p_\theta, q_\phi) + \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x)) \tag{7}$$

We can also simplify $\mathcal{L}(p_\theta, q_\phi)$ as

$$\mathcal{L}(p_\theta, q_\phi) = \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_{phi}(z|x)} \tag{8}$$
$$= \int_z q_\phi(z|x) \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} \tag{9}$$
$$= \int_z q_\phi(z|x) \log \frac{p(z)}{q_\phi(z|x)} + \int_z q_\phi(z|x) \log p_\theta(x|z) \tag{10}$$
$$= \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) - \mathcal{D}_{KL}(q_\phi(z|x)||p(z)). \tag{11}$$

Combining 7 and 11 we have the result.

Corollary 2 *Since $\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$ is bigger than 0, we have that $\log p_\theta(x) \geq \mathcal{L}(p_\theta, q_\phi)$.*

In equation (12) note that x is fixed (given by the data points) and q_ϕ is any distribution. In general q_ϕ may not be a map which assigns high probability to z values which generate likely x. But suppose we are maximizing the left had side of the equation (12). This maximizes the log likelihood function $\log p_\theta(x)$ while also minimizing the KL distance between $\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$. As we discussed in section 3 we cannot compute $p_\theta(z|x)$, so minimizing $\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$, we hope to find a distribution $q_\phi(z|x)$ such that $q_\phi(z|x)$ is *close* to $p_\theta(z|x)$). We assume that the parameter space in the neural network used to calculate $q_\phi(z|x)$ is large enough so that it is possible to find a *match* in parameter space ϕ to unknown distribution $p_\theta(z|x)$). Ideally, if the KL distance is 0 in the LHS of (12), we would be directly maximizing $\log p_\theta(x)$.

Equation (12) is useful because the LHS is the log likelihood which we want to maximize (plus some error term). Since we cannot compute the log likelihood, we maximize the RHS of the (12) instead. One can use stochastic gradient descent to perform the maximization. Hence we have a new objective now, i.e.

$$\max_{\theta,\phi} \; \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) - \mathcal{D}_{KL}(q_\phi(z|x)||p(z)) \tag{12}$$

$$\text{Reconstruction term: } \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) \tag{13}$$

$$\text{Regularization term: } -\mathcal{D}_{KL}(q_\phi(z|x)||p(z))\mathcal{D}_{KL}(q_\phi(z|x)||p(z)) \tag{14}$$

In order to use stochastic gradient descent algorithm to the above function we need to compute the gradient w.r.t θ, ϕ. Let us now see how can we perform the gradient update by simplifying each term separately in the next subsection. For simplicity we would assume a Gaussian prior $p(z)$. Since $\mathcal{Z}$ is a lies in general in a large dimension let $p(z) = \mathcal{N}(0, I)$. We hope that, by our modelling choices, we learn $q_\phi(z|x)$ to lie in a small subset of large space $\mathcal{Z}$ which gives higher probability to areas which generate samples like the data set $X_i, i = 1, \dots, n$.

4.1 Reconstruction term

The reconstruction term is a expectation over distribution q_ϕ, which is the encoder network. One way to do it would be to generate large number of samples from the distribution q_ϕ using the encoder network and then approximate the expectation by a sample average,

$$\mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) \approx \frac{1}{k} \sum_{i=1}^{k} \log p_\theta(x|z_i)$$

. Usually, as is standard in stochastic gradient descent, we take one sample z from the encoder network $q_\phi(z|x)$ and treat $\log p_\theta(x|z)$ as an estimate of $\mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z)$.

4.2 Regularization term

Let us assume that $q_\phi(z|x)$ is a normal distributed with mean as a function of x, z and some parameter ϕ. So,

$$q_\phi(z|x) = \mathcal{N}\left(\mu_\phi(x,z), \Sigma_\phi(x,z)\right) \tag{15}$$

Here, both $\mu_\phi(x)$ and $\Sigma_\phi(x)$ are neural networks with some architecture that depends on ϕ which is learned through data points $X_i, i = 1, \dots, n$. Note that z can be thought as a constant in function $\mu_\phi(x,z), \Sigma_\phi(x,z)$. $\mathcal{D}_{KL}(q_\phi(z|x)||p(z))$ is now a KL divergence between two multivariate Gaussian distribution and can be shown as [1],

$$\mathcal{D}_{KL}\left(\mathcal{N}\left(\mu_\phi(x,z), \Sigma_\phi(x,z)\right)||\mathcal{N}(0,I)\right) \tag{16}$$

$$= \frac{1}{2}\left(\text{Tr}(\Sigma_\phi(x,z)) + \mu_\phi(x,z)^T \mu_\phi(x,z) - \dim(\mathcal{Z}) - \log\det(\Sigma_\phi(x,z))\right) \tag{17}$$

Using the above approximation we can re-write the objective function as

$$\mathcal{L}(p_\theta, q_\phi) \approx \log p_\theta(x|z) - \mathcal{D}_{KL}\left(\mathcal{N}\left(\mu_\phi(x,z), \Sigma_\phi(x,z)\right)||\mathcal{N}(0,I)\right) \tag{18}$$

4.3 Gradient computation

Our objective is to learn the parameters θ, ϕ of the neural network by optimizing (18). After the simplifications given above we are left with a network which can be visualized as below in Figure 7. In this example z lies in a one dimensional space and x lies in a two dimensional space.

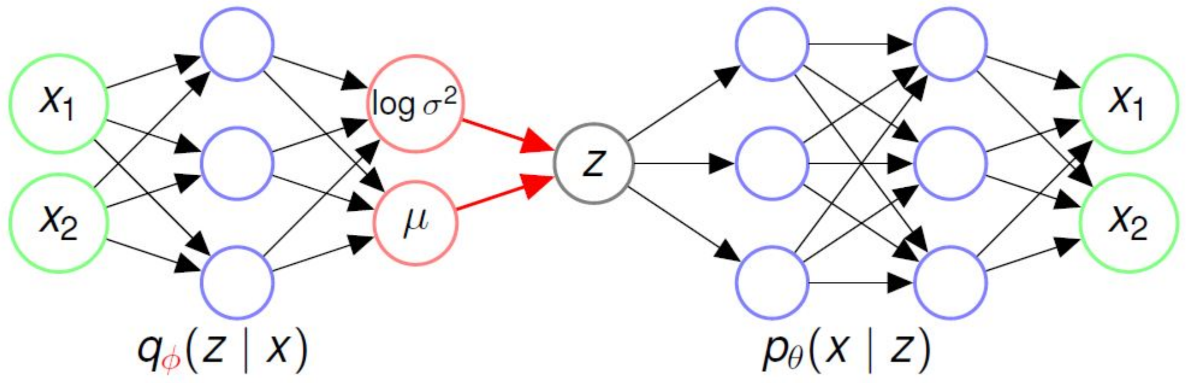


Figure 7: Simplified VAE representing equation (18)

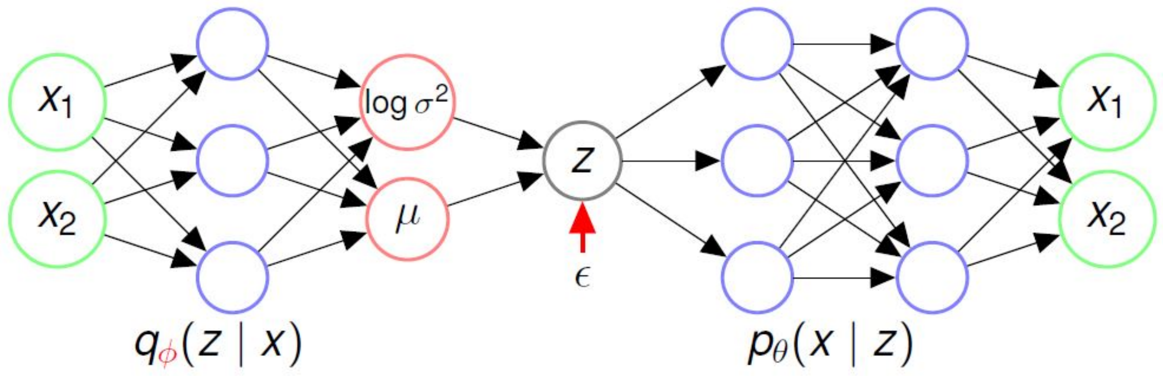


Figure 8: VAE after the reparametrization trick

We need to differentiate the RHS of (18) w.r.t neural network parameters θ, ϕ. Although both terms are look like they can be differentiated, we have missed one detail. The regularization term in 17 is differentiable. The reconstruction step requires us to first sample from distribution $q_\phi(z|x)$ and then compute the function $\log p_\theta(x|z)$. This is a non-continuous operation and has no gradient w.r.t ϕ. So, we cannot perform back propagation through the encoder network (e.g. the red units in Figure 7). To simplify this operation we use a property of Gaussian distribution and write,

$$q_\phi(z|x) = \mathcal{N}\left(\mu_\phi(x,z), \Sigma_\phi(x,z)\right) = \mu_\phi(x,z) + \sqrt{\Sigma_\phi(x,z)} \times \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I). \tag{19}$$

Thus we can write the reconstruction term as,

$$\log p_\theta(x|z) = \log p_\theta(x|z = \mu_\phi(x,z) + \sqrt{\Sigma_\phi(x,z)} \times \epsilon) \tag{20}$$

We can visualize the transformation below in the network in Figure 8.

Now the network takes input as x and ϵ where the distribution of ϵ does not depend on our model parameters ϕ. We can treat the RHS of (20) as deterministic function of x and ϵ and continuous w.r.t. parameter ϕ. So we differentaite (or back propagate through the network) w.r.t ϕ.

The final flowchart of the variational autoencoder is represented in Figure 9 Forward pass: The forward process can be intuitively thought as the following: The encoder network accepts the data

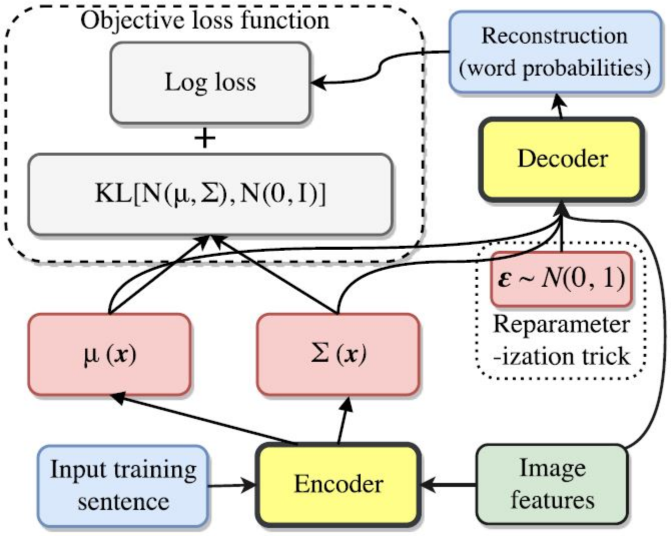


Figure 9: Flowchart for the VAE

points in sequence $X_i, i = 1, \ldots, n$ and computes the mean and variance as function of data points. This combines with the sample ϵ to produce a sample z from latent space $\mathcal{Z}$. z is then fed into the decoder network to reconstruct a sample in space of X. Backpropagation: During the training phase, we have a stream of data points $X_1, \ldots X_n$. We perform one update step of the stochastic gradient descent algorithm on log loss (reconstrction term (15)) plus KL divergence (regularization term (17)) upon arrival of one data point X_i. Each gradient is computed by backpropagation through the network as discussed in section 4.3.

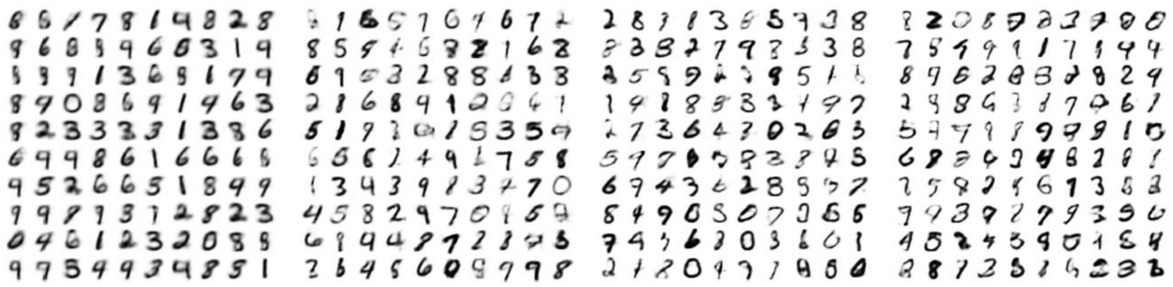
5 Example

Consider the handwritten digit images of $28 * 28$ from the MNIST dataset. Suppose we train a variational autoencoder to learn the lower dimensional latent space in this set of images. After training, say we generate new data points using this trained decoder. Figure 10 shows the artificially generated digits for different dimensions of the latent space. This example is derived from [2]. As we can see that the trained autoencoder with dimension of latent space as 2 does a poor job in generating new images that look like handwritten digits. As we increase the latent space dimension to 20 we can see much clearer images of hand-written digits.

References

[1] C. Doersch. Tutorial on variational autoencoders. *arXiv:1606.05908*, 2014.

[2] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2014.



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Figure 10: Random samples from learned generative models of MNIST for different dimensionalities of latent space. [2]