**CS7140: Advanced Machine Learning** **Spring 2020**

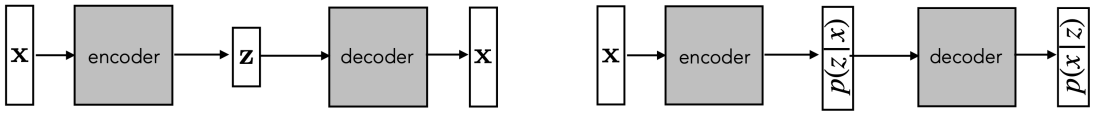
# Lecture 12: Approximate Inference, VAEs

*Lecturer: Rose Yu* *Scribes: Akshay Kurmi, Shuo Jiang, Yash Bhavsar*

## 12.1 Autoencoders

### 12.1.1 Autoencoder

Autoencoders are neural networks that allow us to learn a latent representation of the data. These models are primarily used for two purposes: (1) to perform dimensionality reduction of the input data and (2) for feature learning. An autoencoder consists of an encoder which learns a mapping from the input space to the latent space and a decoder which learns a mapping from the latent to the input space. The latent representation learnt can be either deterministic or stochastic. In a deterministic autoencoder, the encoder takes the input $x$ and maps it to the latent variable $z$, while the decoder takes the latent variable $z$ and maps it back to $x$. Essentially the autoencoder tries to reconstruct the input $x$, but the reconstructed value generated might not be perfect. A stochastic autoencoder, which is an extension of the deterministic version, learns a probabilistic distribution over the latent space. Its architecture is very similar to the deterministic autoencoder, except that the outputs of the encoder and decoder are probability distributions. The encoder takes $x$ as its input, and tries to approximate the probability distribution of the latent variable given the input $p(z|x)$. The decoder takes as input samples from $p(z|x)$ and tries to approximate the probability distribution of the generated output given the latent variable $p(x|z)$. The deterministic and stochastic autoencoders are illustrated in Figure 12.1.




(a) Deterministic Autoencoder (b) Stochastic Autoencoder

Figure 12.1: Types of Autoencoder

Figure 12.2 illustrates an example of a deterministic autoencoder reconstructing handwritten digits. The encoder is implemented using a multi-layer convolutional neural network that maps the input image to a 10 dimensional latent vector $h$. The decoder is implemented using a multi-layer deconvolutional neural network that maps the latent vector to a reconstructed image. The autoencoder is a general framework that can use any kind of neural network architecture to implement the encoder and decoder. The choice of model architecture depends on the type of data you are using and the type of probabilistic distribution you want to learn.

### 12.1.2 Denoising Autoencoder

A denoising autoencoder is a type of autoencoder that takes a version of the data corrupted by some noise as input. The corruption process $C(\tilde{x}|x)$ is a stochastic process that adds noise to the input $x$. The reconstruction distribution is $p(x|\tilde{x})$ and hence the goal here is to reconstruct a denoised version of the input.

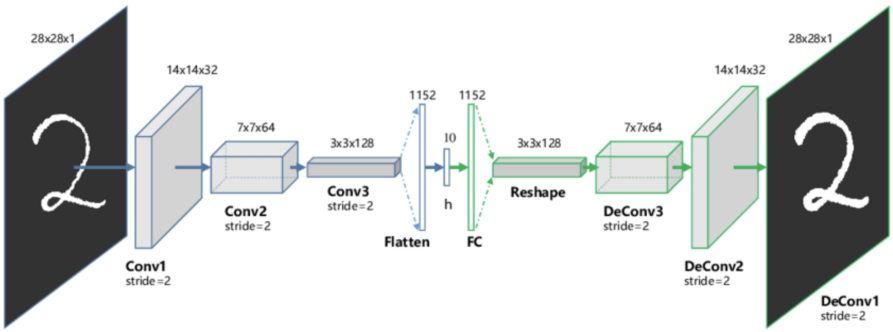



Figure 12.2: Example of Deterministic Autoencoder

The rest of the framework is the same as a vanilla autoencoder. During training, we sample data from the corruption process, and estimate the parameters of the model using gradient based optimization algorithms that try to minimize the reconstruction error $\|g(f(\tilde{x})) - x\|^2$. This is equivalent to performing stochastic gradient descent on the log likelihood distribution $\mathbb{E}_{x\sim p(x)}\mathbb{E}_{\tilde{x}\sim C(\tilde{x}|x)} \log p(x|z)$. A denoising autoencoder is illustrated in Figure 12.3

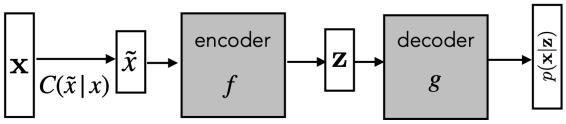



Figure 12.3: Denoising Autoencoder

The dimensionality of the latent space is usually smaller than the dimensionality of the input space. A variant of autoencoders called sparse autoencoders use a latent space larger than the input space.

## 12.2 Variational Autoencoder

### 12.2.1 Approximate Inference

The Variational Autoencoder combines the ideas of an autoencoder and variational inference. Variational inference is a type of approximate inference, that can be used to approximate complex distributions like images for example, which are a complex distribution over pixels. Variational inferences uses the KL-divergence which is also known as relative entropy. The KL-divergence of $Q$ from $P$ is the expected log ratio of distributions $Q$ and $P$ under the distribution $Q$. The KL-divergence is a asymmetric metric i.e. $D_{KL}(Q||P) \neq D_{KL}(P||Q)$. The key technique used in variational inference is the Gibbs inequality.

$$\begin{aligned} D_{KL}(Q||P) &= \sum_x Q(x) \log \frac{Q(x)}{P(x)} \\ &\geq 0 \end{aligned}$$

The Gibbs inequality states that the KL-divergence between two distributions $Q$ and $P$ is always non-negative. This is intuitive because, KL-divergence is a distance metric between two distributions, and distance metrics need to be non-negative. The Gibbs inequality also finds several application in statistical physics, machine learning and data science.

### 12.2.2 Variational Free Energy

Variational inference tries to approximate a complex distribution $P(x)$ with a simple distribution $Q(x;\theta)$. The probability distribution $P(x)$ can be written as

$$P(x) = \frac{1}{Z}P^\star(x) = \frac{1}{Z}\prod_{m=1}^{M}\phi(x_m)$$

where $Z$ is the partition function and $P^\star(x)$ is the unnormalized distribution which can be decomposed as a product of $M$ factors. By Gibbs inequality, we have,

$$\begin{aligned} D_{KL}(Q||P) &= \log Z - \sum_m \mathbb{E}_Q[\log\phi] - H_Q \\ &= \log Z + F[P^\star, Q] \\ &\geq 0 \end{aligned}$$

Here, $F[P^\star, Q] = -\sum_m \mathbb{E}_Q[\log\phi] - H_Q$ is known as variational free energy. In order to approximate $P(x)$ using the distribution $Q(x;\theta)$, we try to minimize the KL-divergence between these two distributions. Because $\log Z$ is a constant, minimizing the relative entropy becomes equivalent to minimizing the variational free energy.

### 12.2.3 ELBO

The variational free energy gives a lower bound on the partition function.

$$\log Z \geq -F[P^\star, Q] = \sum_m \mathbb{E}_Q[\log\phi] + H_Q$$

Given a joint distribution $p(x)$, we can derive the lower bound as follows,

$$\begin{aligned} \log p(x) &= \log \int_z p(x,z)\frac{q(z)}{q(z)}dz \\ &= \log \mathbb{E}_q\left[\frac{p(x,z)}{q(z)}\right] \\ &\geq \mathbb{E}_q\left[\log\frac{p(x,z)}{q(z)}\right] \\ &= \mathbb{E}_q[\log p(x,z)] - \mathbb{E}_q[\log q(z)] \end{aligned}$$

Here, $\mathcal{L} = \mathbb{E}_q[\log p(x,z)] - \mathbb{E}_q[\log q(z)]$ is known as the evidence lower bound or ELBO. The evidence lower bound provides a lower bound on $\log p(x)$ and hence we can use $\mathcal{L}$ to approximately fit the model.

### 12.2.4 Variational Autoencoder (VAE)

Variational Autoencoder tries to use neural networks to model directed graph model. Here we go back to graphical model notation again. As in Figure 12.4, we have observed data $x$ as input and latent variable $z$. We have $N$ independent copies of the variable in the bounding box, and such graphical model is governed by parameter $\theta$. We want to learn such graphical model. Instead of doing traditional graphical model based variation inference approach, or MCMC, we would like to utilize the power of neural networks because they

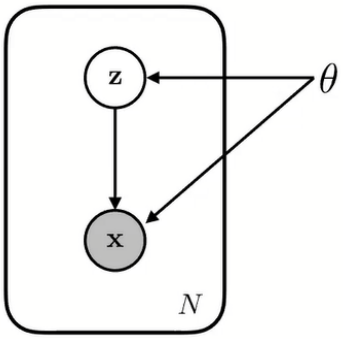



Figure 12.4: Graphical model of VAE

are more flexible and able to approximate broader classes of functions.

As we start from our ELBO, the objective function 12.1, which has a lower bound by a expectation of log of joint distribution and entropy of $q$.

$$\log p(x) \geq \mathbb{E}_q[\log p(x,z)] - \mathbb{E}_q[\log p(z|x)] \tag{12.1}$$

So if we expand such objective function, we will get

$$\begin{aligned}
\log p(x) &\geq \mathbb{E}_q[\log p(x,z)] - \mathbb{E}_q[\log q(z|x)] \\
&= \mathbb{E}_q[\log p(x|z)\, q(z)] - \mathbb{E}_q[\log q(z|x)] \\
&= \mathbb{E}_q[\log p(x|z)\, q(z)] - \mathbb{E}_q[\log q(z|x)] \\
&= \mathbb{E}_q[\log p(x|z)] - \mathbb{E}_q\left[\log\left(\frac{q(z|x)}{p(z)}\right)\right] \\
&= \mathbb{E}_q[\log p(x|z)] - KL[q(z|x)\,||p(z)]
\end{aligned} \tag{12.2}$$

Once we do that, we have a better way to introduce neural networks right now. This is still the evidence lower bound, but just by rearranging the terms. If we approximate $p(x|z)$ as a neural network, this gives us the decoder network, which generate data $x$ from latent code $z$. The $q(z|x)$ can also be modeled as neural network which gives us the encoder distribution, which encodes data $x$ into some latent codes $z$. $p(z)$ is some prior distribution of latent code. Thus we get the structure of variational autoencoder as Figure 12.5. Here observed data $x$ is fed into encoder, encoder outputs inference distribution $q(z|x)$, This allow us to sample random variable $z$, which is the latent code. Then the decoder takes the latent code $z$ and generates data $x$. The question now is how to backpropagate gradient from stochastic random variable. Because sampling distribution is not differentiable, which poses significant challenge.

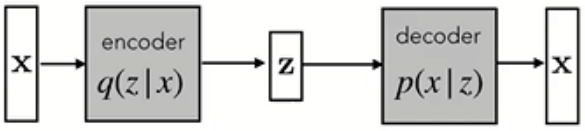



Figure 12.5: Structure of Autoencoder

The trick we use to successfully backprop is called **Reparameterization Trick**. That is the key technique that enables auto differentiation over backprop for VAE. Also this trick is generic technique that allow us to perform backprop for stochastic random variables. Here, we would like to reparameterize random variable

$z$. If we have a random variable $z \sim N(\mu, \sigma)$, the Gaussian has two parameters $\mu$ and $\sigma$. We reparameterize $z$ as a deterministic function $z = \mu + \sigma\epsilon$ and introduces a additional random variable $\epsilon \sim N(0, I)$. Notably, $\mu$ and $\sigma$ are deterministic parameters of Gaussian and the only randomness comes from $\epsilon$. More generally, if we want a random variable $z$, we can introduce another random variable $\epsilon$ with simpler distribution and find a deterministic function $f$ to transform $\epsilon$ to $z$ with some $\omega$. Such $\omega$ contains parameters of distribution and input $x$.

$$z = f(\epsilon, \omega) \tag{12.3}$$

With a graphical demonstration, we have diamond shape for deterministic and circle for random variables


Figure 12.6: Graphical nodes

as in Figure 12.6. In the original computational tree, we have deterministic variables $x$ and parameters $\omega$, and some $z$ is sampled from $q(z|x)$ as shown in Figure 12.7 left part and finally outputs reconstructed value $x$. The problem here is because random variable $z$ is stochastic, and sampling operation is not differentiable, thus there is no way for us to calculate the gradients. Using the reparameterization trick, we are going to transform the computational graph slightly, as in Figure 12.7 right part, we can transform $z$ from a ramdom variable to a deterministic variable. Here the random variable $\epsilon$ is introduced as a root of computational tree then will not cause any problem with backprop. So we can calculate the backward path with gradients $\frac{\partial x}{\partial z}$ and $\frac{\partial z}{\partial \omega}$
The modified architecture of VAE incoperation reparameterization trick is shown in Figure 12.8. Here we still

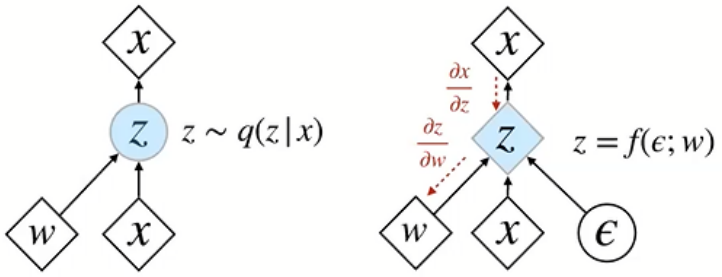
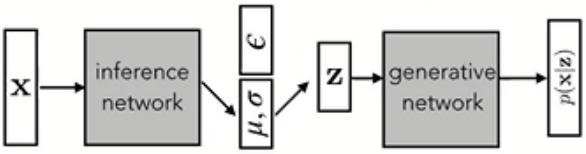



Figure 12.7: Computational graph

have the input $x$, and finally get $p(x|z)$ at the end. Here we call the encoder inference network and decoder generative network. You can see additional random variable $\epsilon$ is introduced to perform reparameterization trick. The parameters $\mu$ and $\sigma$ are outputs of inference network. The neural network automatically finds the proper parameters for us, then we can train the neural network end-to-end with backprop.

VAEs can be implmented in many scenarios, perhaps the most popular applications for VAEs include generating samples and dimension reduction. However VAEs have their own issues. The main issue with VAEs is blurring images that if we use VAE to generate images. In Figure 12.9, we have input images at left side and reconstruction from VAE at right side. You can see reconstructions keep the contour of original images but lose much detail and very blurry. This is usually due to we model the term $\mathbb{E}_q[\log p(x|z)]$ in VAE loss function with $\|\tilde{x} - x\|$ which can be bad for images.




$$z = \mu + \sigma\epsilon \qquad \epsilon \sim \mathcal{N}(0,I)$$

Figure 12.8: VAE architecture


Figure 12.9: VAE image reconstruction

### 12.2.5 Disentangled Representation

Disentangled representation means a single latent unit being sensitive to variations in single generative factors. A variational autoencoder encourages the posterior distribution over the generative factors $q(z|x)$ to be closer to the isotropic Gaussian $N(0, I)$, it promotes disentanglement of the latent generative factors. This is due to the fact that the covariance $\Sigma$ of isotropic Gaussian is equal to an identity matrix $I$ meaning all the dimensions are independent. In the ELBO, this is promoted by the second term representing the disentanglement:

$$\mathcal{L} = \mathbb{E}_{q(z|X)}[\log p(X|z)] - \mathbf{D}_{KL}[q(z|X)||p(z)] \tag{12.4}$$

However, the learning required for effective disentanglement may not be enough since the in variational autoendcoder we also want to autoencode or reconstruct the our input and so the reconstruction loss may be too strong compared to the second term. So we define an additional weight to the second term $\beta$ to create a stronger constraint on the latent bottleneck via $\beta > 1$. So, now the equation looks like this:

$$\mathcal{L} = \mathbb{E}_{q(z|X)}[\log p(X|z)] - \beta\mathbf{D}_{KL}[q(z|X)||p(z)] \qquad \text{(promotes disentanglement)} \tag{12.5}$$

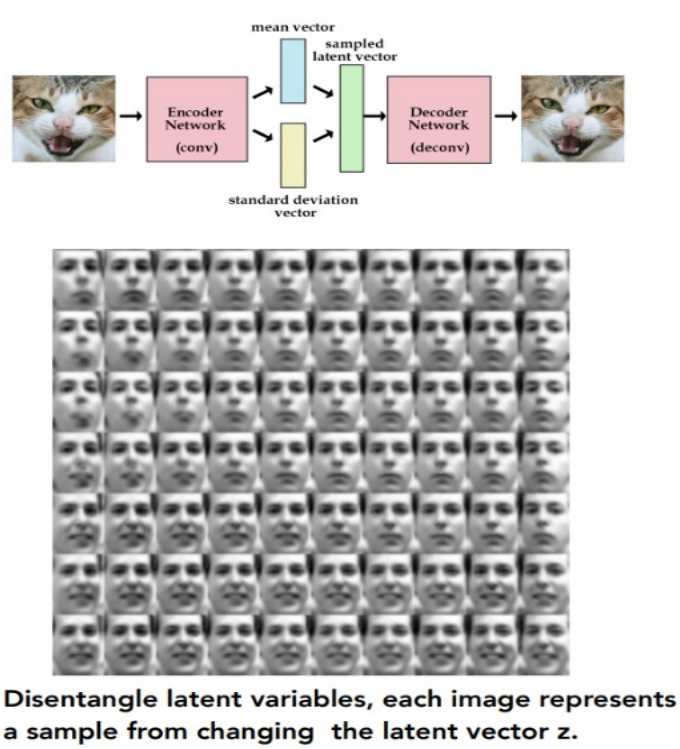



Figure 12.10: Image showing disentanglement

### 12.2.6 Discrete Stochastic Operation

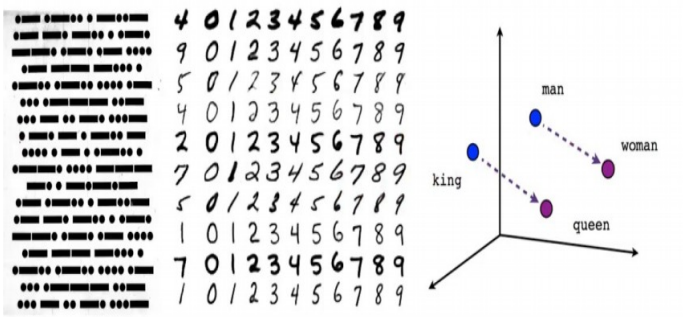



Figure 12.11: Discrete operation

The reparameterization trick does not work for the discrete values or discrete variable is because mathematically it is impossible for a non-degenerate function that maps a continuous set onto a discrete set to be differentiable. That is, for the function relation $z = g(\phi, \epsilon)$, it does not make any sense to conceive $\frac{\partial z}{\partial \phi}$ in the discrete case, regardless of the value of $\epsilon$. Alternatively, it means we cannot backpropagate the gradients through discrete nodes in the computational graph.

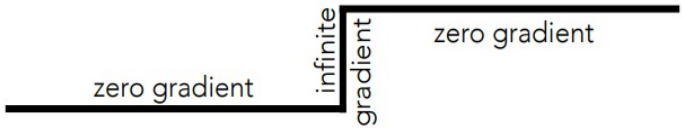



Figure 12.12: Discrete function

### 12.2.7 Gumbel-Soft max

First let us know what gumbel distribution is. The random variable $G$ is said to have a standard Gumbel distribution if $G = -\log(-\log(U))$ with $U \sim Unif[0, 1]$. For us, its importance is a consequence that we can parameterize any discrete distribution in terms of Gumbel random variables by using the following fact:

Let $Z$ be a discrete random variable with $P(Z = k) \propto \pi_k$ random variable and let $\{G_k\}_{k \leq K}$ be an i.i.d sequence of standard Gumbel random variables. Then:

$$Z = arg\max_k(\log \pi_k + G_k) \tag{12.6}$$

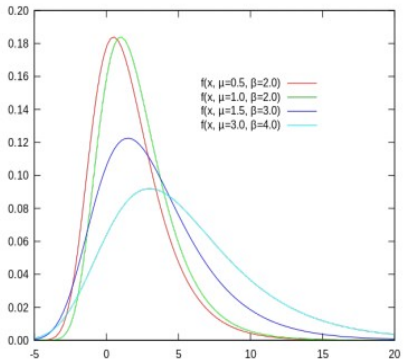



Figure 12.13: Gumbel distribution

Unfortunately, the arg max operation that relates the Gumbel samples, $the\pi_k's$ and the realizations of the discrete distribution is not continuous. One way of circumvent this, is to relax the discrete set by considering random variables taking values in a larger set.

Therefore, a natural way to relax a discrete random variables is by allowing it to take values in the probability simplex. We can do this by using softmax map:

$$\sigma(\pi) = \frac{e^{(\pi_k)}}{\Sigma_k e^{(\pi_k)}} \tag{12.7}$$

With this definition we can define (instead of the discrete valued random variable $Z$) the sequence of the simple valued random variables:

$$Z^\tau = \frac{e^{(\log \pi_k + G_k)/\tau}}{\Sigma_k e^{(\log \pi_k + G_k)/\tau}} \tag{12.8}$$

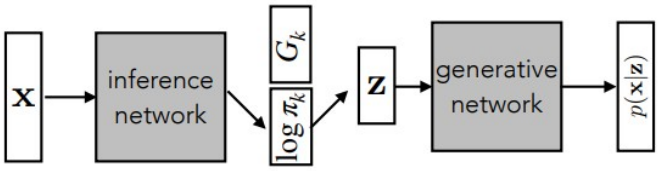



Figure 12.14: Gumbel softmax

The random variable $X^\tau$ defined as above is said to have the concrete distribution, denoted $x^\tau \sim Concrete(\alpha, \tau)$. Example of the effect of the temperature constant $\tau$ on categorical data. As the temperature constant increases the sample becomes more similar to the expectation.

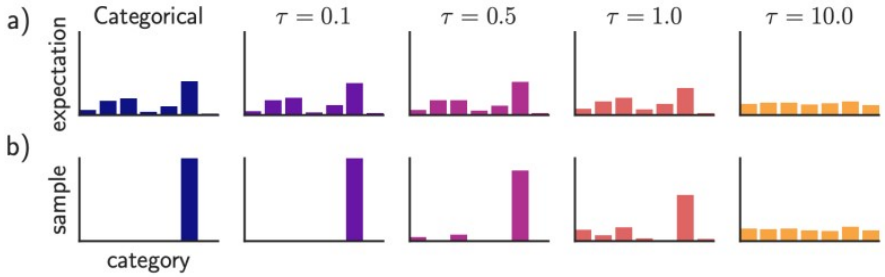



Figure 12.15: Gumbel softmax example