Now we will study how to leverage generative models to sample from a distribution. We will leverage neural networks in the following way:

- First sample a latent variable $z$ from distribution $\mu$, that is easy to sample from. For example, $\mu$ can be the uniform distribution over $[0, 1]$ or the Gaussian distribution.
- Then pass the latent variable through a neural network $g$ and output $g(z)$.

In this lecture, we will cover one of the most popular generative network methods—variational autoencoder (VAE).

**Autoencoder**  Let us first talk about what an autoencoder is. Well, in fact, you have already seen an autoencoder at this point. A special case is just the PCA (and also kernel PCA), which gives the optimal linear encoding/decoding: Given $X = USV^\top$ and and $k \leq r$,

$$
\min_{E \in \mathbb{R}^{d \times k}, D \in \mathbb{R}^{k \times d}} \| X - XED \|_F^2 = \| X - XV_kV_k^\top \|_F^2
$$

But we can also have encoders and decoders that are not linear mappings. Let encoders $\mathcal{E}$ and decoders $\mathcal{D}$ denote families of deep networks from $\mathbb{R}^d$ to $\mathbb{R}^k$ and from $\mathbb{R}^k$ to $\mathbb{R}^d$

$$
\min_{f \in \mathcal{E}, g \in \mathcal{D}} \sum_{i=1}^{n} \| x_i - g(f(x_i)) \|_2^2
$$

This is called an autoencoder, which deterministically map each example $x_i$ to a latent code $z_i$, back to some approximation of $x_i$. We say that $\mathbb{R}^k$ is the latent space, and $f(x) \in \mathbb{R}^k$ is latent representation of $x$.

**Variational Autoencoder (VAE)**

We will now leverage the idea of autoencoder to build generative models. Intuitively, we should take the decoder $g$ from an autoencoder as our generative network, which is a mapping from a low-dimensional latent space $\mathbb{R}^k$ to the example space $\mathbb{R}^d$.

In particular, suppose we have a sample $x_1, \ldots, x_n$ drawn from some distribution $P$. We want to find $g$ so that $g(z_i) \approx x_i$ for each $i$, where each $z_i$ is drawn from a Gaussian distribution. VAE construct a distribution for each $z_i$ based on each $x_i$. The method runs over iterations, and in each iteration does the following:
1. Encode each example into Gaussian mean-variance parameters \((\mu_i, \Sigma_i) \leftarrow f(x_i)\).

2. Sample latent variable from Gaussian: \(z_i \sim N(\mu_i, \Sigma_i)\).

3. Decode \(\hat{x}_i = g(z_i)\).

4. Taking a gradient descent step (or any other optimization method) to further minimize the VAE objective

\[
\sum_{i=1}^{n} \ell(x_i, \hat{x}_i) + \lambda \text{KL} \left( N(\mu_i, \sigma_i^2 I), N(0, I) \right)
\]

where \(\ell(x_i, \hat{x}_i)\) is “reconstruction error”. For example, \(\ell(x_i, \hat{x}_i) = \|x_i - \hat{x}_i\|^2\). We will go into the details of the gradient update step in a bit.

In the VAE objective, KL denotes KL divergence: for any two distributions \(p\) and \(q\),

\[
\text{KL}(p||q) = \int p(z) \ln \frac{p(z)}{q(z)} dz
\]

KL divergence is a dissimilarity measure between distributions, with two important properties:

- \(\text{KL}(p||q) \geq 0\) for any \(p, q\).
- \(\text{KL}(p||q) = \text{KL}(q||p)\) if and only if \(p = q\).

KL divergence encourages the individual distributions \(N(\mu_i, \Sigma_i)\) to be close to the distribution \(N(0, I)\). This is useful because \(N(0, I)\) is the “source” distribution for the generative models—that is, we output \(g(z)\) with \(z \sim N(0, I)\). The smaller the KL divergence is, the closer this sampling has to approximate the training distribution.

**Derivation from Variational Inference**

VAE is based on ideas from variatioinal inference (VI), which is a popular method to perform approximate inference in probabilistic models. We won’t get into the details of VI here, but we will discuss the relevant ideas that lead to VAE. Let \(\mathcal{P} = \{p_\theta \mid \theta \in \Theta\}\) be a family of probability distributions over observed and latent variables \(x\) and \(z\). Given a set of observed variables \(S = \{x_1, \ldots, x_n\}\), we would like to find a distribution in \(\mathcal{P}\) to minimize:

\[
\min_{p \in \mathcal{P}} \text{KL}(\hat{p}_S||p) = \min_{p \in \mathcal{P}} \sum_{x \in S} \hat{p}_S(x) \ln \frac{\hat{p}_S(x)}{p(x)}
\]

where \(\hat{p}_S\) denotes the empirical distribution over the data set. Note that \(\sum_{x \in S} \hat{p}_S(x) \ln p_S(x)\) does not depend on the choice of \(p\). Thus, the minimization is equivalent to the following maximization problem:

\[
\max_{p \in \mathcal{P}} \sum_{x \in S} \hat{p}_S(x) \ln p(x) \iff \max_{p \in \mathcal{P}} \sum_{x_i \in S} \ln p(x_i) \iff \max_{p \in \mathcal{P}} \sum_{x_i \in S} \int p(x_i, z) dz
\]
Thus, minimizing the KL divergence objective is the same as maximizing log-likelihood. The problem above is typically intractable for generative models with high-dimensional \(z\), since it involves computing an integral over all \(z\)’s.

To circumvent the intractability, the VI method aims to optimize a tractable lower bound of the log-likelihood. To do that, we introduce a family of approximate distributions \(Q = \{q_\gamma \mid \gamma \in \Gamma\}\). (Each distribution \(q\) is parameterized by \(\gamma\).) Observe that for any fixed \(x\),

\[
\ln p(x) = \int q(z|x) \ln p(x) \, dz \\
= \int q(z|x) \ln \frac{p(x)q(z|x)p(z|x)}{p(z|x)q(z|x)} \, dz \\
= \int q(z|x) \ln \frac{q(z|x)}{p(z|x)} \, dz + \int q(z|x) \ln \frac{p(x,z)}{q(z|x)} \, dz \\
= \text{KL} \left( q(z|x) \middle\| p(z|x) \right) + \text{ELBO} \\
\geq 0
\]

As indicated above, the KL term is always non-negative, and so the second term is a lower bound for \(\ln p(x)\). The second term is hence called the evidence lower bound (ELBO). For any two distributions \(p_\theta \in P\) and \(q_\gamma \in Q\), let us write

\[
\text{ELBO}(x; \theta, \gamma) = \int q(z|x) \ln \frac{p_\theta(x,z)}{q_\gamma(z|x)} \, dz
\]

The VI method then uses gradient-based method to optimize the objective

\[
\max_{\theta} \sum_{x_i \in S} \max_{\gamma_i} \mathbb{E}_{q_{\gamma_i}(z|x_i)} \left[ \ln \frac{p_\theta(x_i, z)}{q_{\gamma_i}(z|x_i)} \right]. \tag{1}
\]

In each iteration, we do two-step update:

1. First, for each example \(i\): update \(\gamma_i\)
   \[
   \gamma_i \leftarrow \gamma_i + \eta_{\gamma} \tilde{\nabla}_{\gamma} \text{ELBO}(x_i; \theta, \gamma^{(i)}), \tag{2}
   \]

2. Update \(\theta\)
   \[
   \theta \leftarrow \theta + \eta_\theta \tilde{\nabla}_{\theta} \sum_i \text{ELBO}(x^{(i)}; \theta, \gamma^{(i)}), \tag{3}
   \]

where \(\tilde{\nabla}\) denote unbiased estimate for the gradients and \(\eta_{\gamma}\) and \(\eta_\theta\) are the learning rates.
Reparameterization trick. To estimate the gradient $\nabla \text{ELBO}(x; \theta, \gamma) = \nabla_\gamma \mathbb{E}_{q_\gamma(z|x)} \left[ \log \frac{p_\theta(x,z)}{q_\gamma(z|x)} \right]$, we will leverage a reparameterization trick. Let us introduce a fixed, auxiliary distribution $\nu(\epsilon)$ and a differentiable function $T(\epsilon; \gamma)$ such that sampling from $q_\gamma(z|x)$ is identical to

$$\epsilon \sim \nu \quad z \sim T(\epsilon; \gamma)$$

Then the gradient computation can be rewritten as:

$$\nabla_\gamma \mathbb{E}_{q_\gamma(z|x)} \left[ \log \frac{p_\theta(x,z)}{q_\gamma(z|x)} \right] = \mathbb{E}_\nu \left[ \nabla_\gamma \log \frac{p_\theta(x,T(\epsilon; \gamma))}{q_\gamma(T(\epsilon; \gamma))} \right] \quad (4)$$

We can then approximate the right hand side of (4) by drawing $\epsilon_1, \ldots, \epsilon_m$ from $\nu$, and then compute the average gradient:

$$\frac{1}{m} \sum_{i=1}^{m} \left[ \nabla_\gamma \log \frac{p_\theta(x,T(\epsilon_i; \gamma))}{q_\gamma(T(\epsilon_i; \gamma))} \right]$$

This is also called Monte Carlo sampling. Note that the gradient $\nabla_\theta \text{ELBO}(x; \theta, \gamma)$ can be estimated with Monte Carlo sampling, but without the reparametrization trick: draw $z_1, \ldots, z_m$ i.i.d. from $p(z|x)$, and then compute the average gradient

$$\frac{1}{m} \sum_{i=1}^{m} \left[ \nabla_\theta \log \frac{p_\theta(x,z_i)}{q_\gamma(z_i|x)} \right]$$

where $\Sigma^{1/2}$ is the Cholesky decomposition of $\Sigma$.

Instantiation via neural nets. Now we will obtain VAE from this framework of VI by instantiating the distributions $p$ and $q$ through neural networks and Gaussian distributions. First, we will have the latent distribution as

$$p_\theta(z) = \mathcal{N}(0, I)$$

Note that this “prior” distribution doesn’t depend on $\theta$. The conditional distribution $p_\theta(x|z)$ corresponds to the decoder. A typical choice is a Gaussian distribution

$$p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$$

where the mean and covariance parameters $\mu_\theta(z), \Sigma_\theta(z)$ are given by a neural network. If $\Sigma_\theta(z) = \sigma^2 I$, then ELBO becomes the VAE objective with squared error as the reconstruction error, that is

$$\ell(x_i, \hat{x}_i) = \|x_i - \hat{x}_i\|_2^2$$

For the approximate distribution $q$, we will have

$$q_\gamma(z|x_i) = \mathcal{N}(\mu(x_i), \Sigma(x_i)),$$

where the parameter $\gamma_i = (\mu(x_i), \Sigma(x_i))$ are mean and covariance parameters given by the encoder neural network. To apply the reparameterization trick, we will have $\nu = \mathcal{N}(0, I)$ and $T(\epsilon; \gamma) = \mu + \Sigma^{1/2}\epsilon$, where $\Sigma^{1/2}$ is the Cholesky decomposition of $\Sigma$. For $\Sigma = \sigma^2 I$, we will simply have $\Sigma^{1/2} = \sigma I$. 
