ELEG 5491: Introduction to Deep Learning

Varational Autoencoder

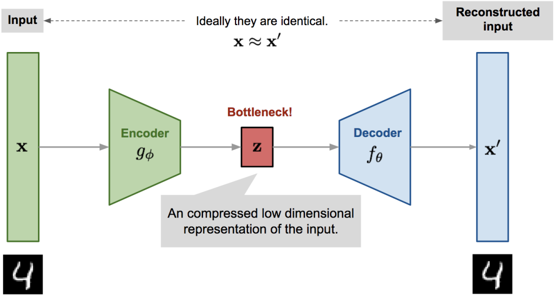
Prof. LI Hongsheng

Office: SHB 428
e-mail: hsli@ee.cuhk.edu.hk
web: https://dl.ee.cuhk.edu.hk

Department of Electronic Engineering
The Chinese University of Hong Kong

March 2023

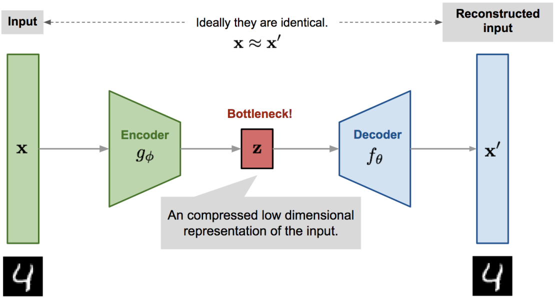
- Autoencoder is a neural network designed to learn an identity function in an unsupervised way to reconstruct the original input while compressing the data in the process so as to discover a more efficient and compressed representation
- **Encoder network** translates the original high-dimension input into the latent low-dimensional code. The input size is larger than the output size
- **Decoder network** recovers the data from the code, likely with larger and larger output layers



The revisit of autoencoder

- The model contains an encoder function $g_\phi(\cdot)$ parameterized by ϕ and a decoder function $f_\theta(\cdot)$ parameterized by θ
- The low-dimensional code learned for input $\mathbf{x}$ in the bottleneck layer is $\mathbf{z}$ and the reconstructed input is $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$
- One common choice is to use the MSE loss for supervision

$$L_{\mathrm{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)} - f_\theta \left(g_\phi \left(\mathbf{x}^{(i)} \right) \right) \right)^2$$



Variational Autoencoder (VAE)

- **Variational Autoencoder** is actually less similar to the conventional autoencoder above, but deeply rooted in variational bayesian and graphical model
- Instead of mapping the input $\mathbf{x}$ into a fixed vector, we would like to map it into a distribution $p_\theta(\mathbf{z})$, parameterized by θ
- Notation
  - Input $\mathbf{x}$
  - Prior $p_\theta(\mathbf{z})$. The assumption of VAE is that $\mathbf{z} \sim \mathcal{N}(0, 1)$.
  - Likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$
  - Posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$
- Assuming that we know the real parameter θ^* for this distribution. To generate a sample that looks like a real data point $\mathbf{x}^{(i)}$, we following the following steps
  - First, sample a $\mathbf{z}^{(i)}$ from a prior distribution $p_{\theta^*}(\mathbf{z})$
  - Then a value $\mathbf{x}^{(i)}$ is generated from a conditional distribution $p_{\theta^*}\left(\mathbf{x} \mid \mathbf{z} = \mathbf{z}^{(i)}\right)$
- The optimal θ^* can be obtained via maximizing the log likelihood of all training samples

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)})$$

Variational Autoencoder (VAE)

- If we involve the latent vector, we will have

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z})p_\theta(\mathbf{z})dz$$

- However, it is impractical to compute $p_\theta(\mathbf{x}^{(i)})$ in this way, as summing up all possible values of $\mathbf{z}$ is untractable
- We introduce a new approximation function to output what is a likely latent code given an input $\mathbf{x}$, $\mathbf{q}_\phi(\mathbf{z}|\mathbf{x})$, parameterized by ϕ

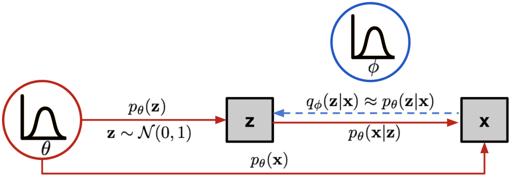


Figure: The dashed line indicates the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

Now the structure looks a lot like an autoencoder:

- The conditional probability $p_\theta(\mathbf{z}|\mathbf{x})$ defines a generative model, similar to the decoder $f_\theta(\mathbf{x}|\mathbf{z})$ in the conventional autoencoder. $p_\theta(\mathbf{z}|\mathbf{x})$ is known as the *probabilistic decoder*
- The approximation function $q_\phi(\mathbf{z}|\mathbf{x})$ is the *probabilistic encoder*, playing a similar role as $g_\phi(\mathbf{z}|\mathbf{x})$ above

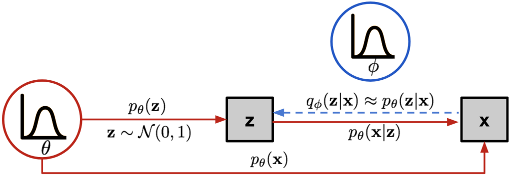


Figure: The dashed line indicates the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

Loss Function: ELBO

- The estimated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ should be very close to the real one $p_\theta(\mathbf{z} \mid \mathbf{x})$
- We can use Kullback-Leibler divergence to quantify the distance between these two distributions. KL divergence $D_{\mathrm{KL}}(X|Y)$ measures how much information is lost if the distribution Y is used to represent X
- In our case, we would like to minimize $D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})\right)$ with respect to ϕ
- Why use $D_{\mathrm{KL}}\left(q_\phi|p_\theta\right)$ (reversed KL) instead of $D_{\mathrm{KL}}\left(p_\theta|q_\phi\right)$

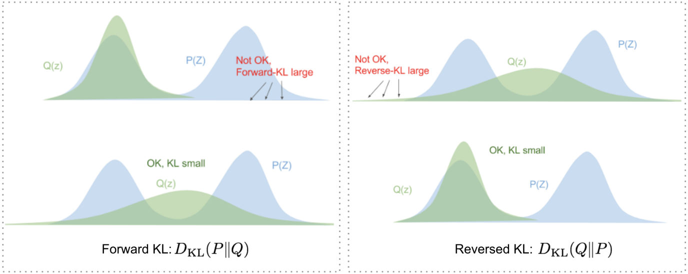


Figure: Forward and reversed KL divergence have different demands on how to match two distributions.

Loss Function: ELBO

- Reverse KL divergence: $D_{\mathrm{KL}}(Q|P) = \mathbb{E}_{z\sim Q(z)} \log \frac{Q(z)}{P(z)}$; minimizing the reversed KL divergence squeezes the $Q(z)$ under $P(z)$
- We expand the equation

$$
\begin{aligned}
& D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})\right) \\
& = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
& = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \\
& = \int q_\phi(\mathbf{z}|\mathbf{x}) \left(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})}\right) d\mathbf{z} \\
& = \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \\
& = \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} \\
& = \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim q\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})\right] \\
& = \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})\right) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})
\end{aligned}
$$

$$D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})\right) = \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})\right) - \mathbb{E}_{\mathbf{z}\sim q\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x} \mid \mathbf{z})$$
$$\log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})\right) = \mathbb{E}_{\mathbf{z}\sim q\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})\right)$$

- The LHS is what we want to maximize when learning the true distributions:
  - To maximize the (log-)likelihood of generating real data ($p_\theta(\mathbf{x})$)
  - To minimize the difference between the real and estimated posterior distributions (D_{KL} works like a regularizer)
- The loss function can be defined as

$$\begin{aligned} L_{\mathrm{VAE}}(\theta, \phi) &= -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})\right) \\ &= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})\right) \\ &= L_{\mathrm{recon}} + L_{\mathrm{KL}} \\ \theta^*, \phi^* &= \arg\min_{\theta,\phi} L_{\mathrm{VAE}} \end{aligned}$$

- Both the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ are modeled as neural networks (e.g., MLPs)
- In Variational Bayesian methods, this loss function is known as the *variational lower bound*, or *evidence lower bound*

Loss Function: ELBO

- *Lower bound* because KL divergence is always non-negative and thus $-L_{\text{VAE}}$ is the lower bound of $\log p_\theta(\mathbf{x})$

$$-L_{\text{VAE}} = \log p_\theta(\mathbf{x}) - D_{\text{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \,\|\, p_\theta(\mathbf{z}|\mathbf{x})\right) \leq \log p_\theta(\mathbf{x})$$

- By minimizing the loss, we are maximizing the lower bound of the probability of generating real data samples
- The expectation term in the loss function invokes generating samples from $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$
- Sampling is a stochastic process and therefore we cannot backpropagate the gradient
- To make it trainable, the reparameterization trick is introduced: It is often possible to express the random variable $\mathbf{z}$ as a deterministic variable
- For example, a common choice of the form of $q_\phi(\mathbf{z}|\mathbf{x})$ is a multivariate Gaussian with a diagonal covariance structure

$$\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}),$$

where $\odot$ refers to element-wise product

- The random variable $\mathbf{z}$ is expressed as a deterministic variable $\mathbf{z} = \mathcal{T}_\phi(\mathbf{x}, \boldsymbol{\epsilon})$ where ϵ is an auxiliary independent random variable, and the transformation function $\mathcal{T}_\phi$ (parameterized by ϕ) converts ϵ to $\mathbf{z}$
- The gradients can then be back-propagated to ϕ ($\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ following the multivariate Gaussian assumption)

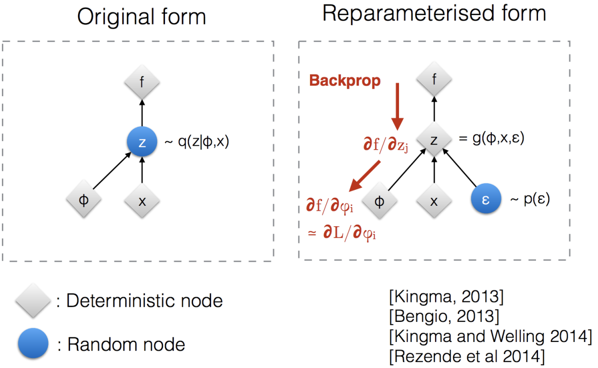


Figure: Illustration of how the reparameterization trick makes the sampling process trainable.(Image source: Slide 12 in Kingma's NIPS 2015 workshop talk

- The first term $L_{\text{recon}} = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x} \mid \mathbf{z})$ is the reconstruction loss
- It can either choose L2 loss or binary cross-entropy loss
- For L2 loss

$$L_{\text{recon}} = \frac{1}{2}\sum_{i=1}^{m} \|\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}\|_2^2$$

- For BCE loss

$$L_{\text{recon}} = -\sum_{i=1}^{m}\sum_{j=1}^{\dim} \mathbf{x}_j^{(i)} \log(\hat{\mathbf{x}}_j^{(i)})$$

- The second term $L_{\text{KL}} = D_{\text{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})\right)$ regularizes the encoded latent vector $\mathbf{z}$ to be close to a standard normal distribution as much as possible
- We derive the loss function with single-variate Gaussian distribution. We formulate

$$p(z) \to \frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(z-\mu_p)^2}{2\sigma_p^2}\right)$$

$$q_\phi(z|\mathbf{x}) \to \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right)$$

Loss Function: ELBO

$$
\begin{aligned}
& -D_{KL}\left(q_\phi\left(z|\mathbf{x}\right)\|p(z)\right) \\
& \quad = \int \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right) \log\left(\frac{\frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(z-\mu_p)^2}{2\sigma_p^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right)}\right) dz \\
& = \int \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right) \times \\
& \quad \left\{-\frac{1}{2}\log(2\pi) - \log\left(\sigma_p\right) - \frac{(z-\mu_p)^2}{2\sigma_p^2} + \frac{1}{2}\log(2\pi) + \log\left(\sigma_q\right) + \frac{(z-\mu_q)^2}{2\sigma_q^2}\right\} dz \\
& = \frac{1}{\sqrt{2\pi\sigma_q^2}} \int \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right) \left\{-\log\left(\sigma_p\right) - \frac{(z-\mu_p)^2}{2\sigma_p^2} + \log\left(\sigma_q\right) + \frac{(z-\mu_q)^2}{2\sigma_q^2}\right\} dz
\end{aligned}
$$

Loss Function: ELBO

Express the above as an expectation, we have

$$
\begin{aligned}
-D_{KL}\left(q_\phi\left(z|\mathbf{x}\right)\|p(z)\right) &= \mathbb{E}_{z\sim q}\left\{\log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{\left(z-\mu_p\right)^2}{2\sigma_p^2} + \frac{\left(z-\mu_q\right)^2}{2\sigma_q^2}\right\} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) + \mathbb{E}_{z\sim q}\left\{-\frac{\left(z-\mu_p\right)^2}{2\sigma_p^2} + \frac{\left(z-\mu_q\right)^2}{2\sigma_q^2}\right\} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\mathbb{E}_{z\sim q}\left\{\left(z-\mu_p\right)^2\right\} + \frac{1}{2\sigma_q^2}\mathbb{E}_{z\sim q}\left\{\left(z-\mu_q\right)^2\right\} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\mathbb{E}_{z\sim q}\left\{\left(z-\mu_p\right)^2\right\} + \frac{\sigma_q^2}{2\sigma_q^2} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\mathbb{E}_{z\sim q}\left\{\left(z-\mu_p\right)^2\right\} + \frac{1}{2} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\mathbb{E}_{z\sim q}\left\{\left(z-\mu_q+\mu_q-\mu_p\right)^2\right\} + \frac{1}{2} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\left[\mathbb{E}_{z\sim q}\left\{\left(z-\mu_q\right)^2\right\} + 2\mathbb{E}_{z\sim q}\left\{\left(z-\mu_q\right)\left(\mu_q-\mu_p\right)\right\}\right. \\
&\qquad \left. + \mathbb{E}_{z\sim q}\left\{\left(\mu_q-\mu_p\right)^2\right\}\right] + \frac{1}{2} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\left[\sigma_q^2 + 2\times 0\times\left(\mu_q-\mu_p\right) + \left(\mu_q-\mu_p\right)^2\right] + \frac{1}{2} \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{\sigma_q^2+\left(\mu_q-\mu_p\right)^2}{2\sigma_p^2} + \frac{1}{2}
\end{aligned}
$$

Substitute $\sigma_p = 1$ and $\mu_p = 0$, we obtain

$$
\begin{aligned}
&-D_{KL}\left(q_\phi\left(z|\mathbf{x}\right)\|p(z)\right) = \log\left(\sigma_q\right) - \frac{\sigma_q^2+\mu_q^2}{2} + \frac{1}{2} \\
&= \frac{1}{2}\log\left(\sigma_q^2\right) - \frac{\sigma_q^2+\mu_q^2}{2} + \frac{1}{2} = \frac{1}{2}\left[1+\log\left(\sigma_q^2\right) - \sigma_q^2 - \mu_q^2\right]
\end{aligned}
$$

KL Loss

- The above loss is defined on only one sample and z is a scalar
- Considering a mini-batch of m samples, the KL loss can be formulated as

$$L_{\mathrm{KL}} = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{\dim} \left[1 + \log(\sigma_j^{(i)})^2 - (\sigma_j^{(i)})^2 - (\mu_j^{(i)})^2 \right]$$

We drop the subscript q here for brevity

Actual Implementation

Model

```
class Model(nn.Module):
    def __init__(self, Encoder, Decoder):
        super(Model, self).__init__()
        self.Encoder = Encoder
        self.Decoder = Decoder

    def reparameterization(self, mean, var):
        epsilon = torch.randn_like(var).to(DEVICE)
        z = mean + var*epsilon
        return z


    def forward(self, x):
        mean, log_var = self.Encoder(x)
        z = self.reparameterization(mean, torch.exp(0.5 * log_var))
        x_hat            = self.Decoder(z)

        return x_hat, mean, log_var
```

Loss

```
from torch.optim import Adam

BCE_loss = nn.BCELoss()

def loss_function(x, x_hat, mean, log_var):
    reproduction_loss = nn.functional.binary_cross_entropy(x_hat, x, reduction='sum')
    KLD      = - 0.5 * torch.sum(1+ log_var - mean.pow(2) - log_var.exp())

    return reproduction_loss + KLD


optimizer = Adam(model.parameters(), lr=lr)
```

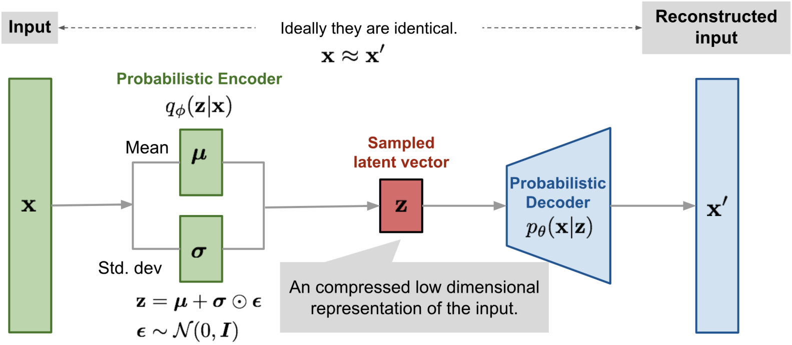


Figure: Illustration of variational autoencoder model with the multivariate Gaussian assumption.

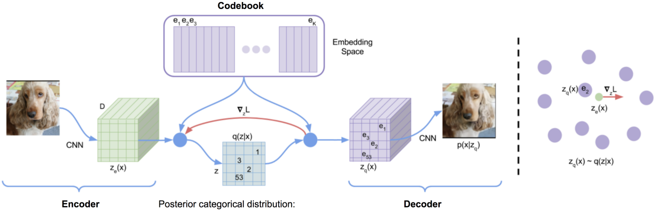
- The original VAE has limited performance. When trained on the CelebA dataset:

Figure: Newly sampled images from VAE trained on CelebA dataset.

- The VQ-VAE model learns a discrete set of latent variables by the encoder
- Vector quantisation map the input vector into a finite set of "code" vectors
- Let $e \in \mathbb{R}^{K \times D}$ for $i = 1, \dots, K$ be the latent code vectors in a codebook of VQ-VAE. The individual code vector is denoted as $e_i \in \mathbb{R}^D, i = 1, \dots, K$
- The encoder output $E(\mathbf{x}) = \mathbf{z}_e$ goes through a nearest-neighbor lookup to match to one of K codes and then this matched code vector becomes the input for the decoder $D(\cdot)$

$$\mathbf{z}_q(\mathbf{x}) = \text{Quantize}(E(\mathbf{x})) = e_k, \text{ where } k = \arg\min_i \|E(\mathbf{x}) - \mathbf{e}_i\|_2$$

- Note that the discrete latent variables can have different shapes in differnet applications: 1D for speech, 2D for image and 3D for video



Posterior categorical distribution:

$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$

VQ-VAE

- Because $\text{argmin}()$ is non-differentiable on a discrete space, the gradients $\nabla_{\mathbf{z}} L$ from decoder inputs $\mathbf{z}_q$ is copied to the encoder output $E(\mathbf{x})$. Other than the reconstruction loss, VQ-VQE also optimizes

$$L = \underbrace{\|\mathbf{x} - D(\mathbf{e}_k)\|_2^2}_{\text{Reconstruction loss}} + \underbrace{\|\text{sg}[E(\mathbf{x})] - \mathbf{e}_k\|_2^2}_{\text{VQ loss}} + \underbrace{\beta \|E(\mathbf{x}) - \text{sg}[\mathbf{e}_k]\|_2^2}_{\text{Commitment loss}}$$

  where $\text{sq}[\cdot]$ is the `stop_gradient` operator
- *VQ loss:* The L2 error between the embedding space (codebook) and the encoder outputs
- *Commitment loss: A measure to encourage the encoder output to stay close to the embedding space and to prevent it from fluctuating too frequently from one code vector to another*
- The code vector in the codebook is updated through exponential moving average (EMA), similar to that in the optimizers
- Given a code vector $\mathbf{e}_i$, if we have n_i encoder output vectors $\{\mathbf{z}_{i,j}\}_{j=1}^{n_i}$ that are quantized to $\mathbf{e}_i$:

$$N_i^{(t)} = \gamma N_i^{(t-1)} + (1-\gamma) n_i^{(t)}, \quad \mathbf{m}_i^{(t)} = \gamma \mathbf{m}_i^{(t-1)} + (1-\gamma) \sum_{j=1}^{n_i^{(t)}} \mathbf{z}_{i,j}^{(t)}, \quad \mathbf{e}_i^{(t)} = \mathbf{m}_i^{(t)} / N_i^{(t)}$$

VQ-VAE

VQ-VAE shows much better performance than the vanilla VAE

Figure: (Left) Training images. (Right) Reconstructions from a VQ-VAE with a $32 \times 32 \times 1$ latent space, with $K = 512$.

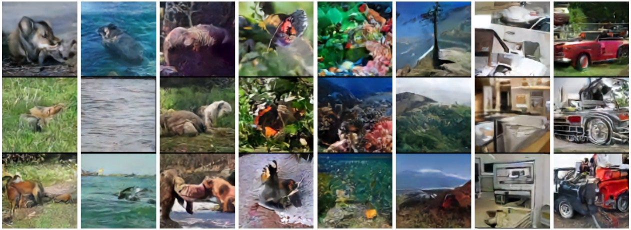


Figure: Samples (128×128) from a VQ-VAE with a PixelCNN prior trained on ImageNet images. Left to right: kit fox, gray whale, brown bear, admiral (butterfly), coral reef, alp, microwave, pickup.