From Autoencoder to Beta-VAE

Aug 12, 2018 by Lilian Weng autoencoder generative-mode

> Autocoders are a family of neural network models aiming to learn compressed latent variab high-dimensional data. Starting from the basic autocoder model, this post reviews se variations, including denoising, sparse, and contractive autoencoders, and then Variat Autoencoder (VAE) and its modification beta-VA

[Updated on 2019-07-18: add a section on VQ-VAE & VQ-VAE-2 .]
[Updated on 2019-07-26: add a section on TD-VAE .]

Autocoder is invented to reconstruct high-dimensional data using a neural network model with a r bottleneck layer in the middle (oops, this is probably not true Variational Autoencode, and we will investigate it in details in later sections). A nice byproduct is dimension reduction: the bottleneck captures a compressed latent encoding. Such a low-dimensional representation can be used a embedding vector in various applications (i.e. search), help data compression, or reveal the under data generative factors

Notation

Symbol	Mean
$\mathcal{D}$	The dataset, $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$, contains n data samples; $\lvert\mathcal{D}\rvert = n$.

Symbol	Mean
$\mathbf{x}^{(i)}$	Each data point is a vector of d dimensions, $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)}]$.
$\mathbf{x}$	One data sample from the dataset, $\mathbf{x} \in \mathcal{D}$.
$\mathbf{x}'$	The reconstructed version of $\mathbf{x}$.
$\tilde{\mathbf{x}}$	The corrupted version of $\mathbf{x}$.
$\mathbf{z}$	The compressed code learned in the bottleneck lay
$a_j^{(l)}$	The activation function for the j-th neuron in the l-th hidden layer
$g_\phi(.)$	The **encoding** function parameterized by ϕ.
$f_\theta(.)$	The **decoding** function parameterized by θ.
$q_\phi(\mathbf{z}\vert\mathbf{x})$	Estimated posterior probability function, also known **probabilistic encoder**.
$p_\theta(\mathbf{x}\vert\mathbf{z})$	Likelihood of generating true data sample given the latent code, also know **probabilistic decoder**.

Autoencoder

Autoencoder is a neural network designed to learn an identity function in an unsupervised wa reconstruct the original input while compressing the data in the process so as to discover a i efficient and compressed representation. The idea was originate the 1980s, and later promoted by the seminal paper by Hinton & Salakhutdinov, 200.

It consists of two network

- *Encoder* network: It translates the original high-dimension input into the latent low-dimensional The input size is larger than the output siz
- *Decoder* network: The decoder network recovers the data from the code, likely with larger and la output layers

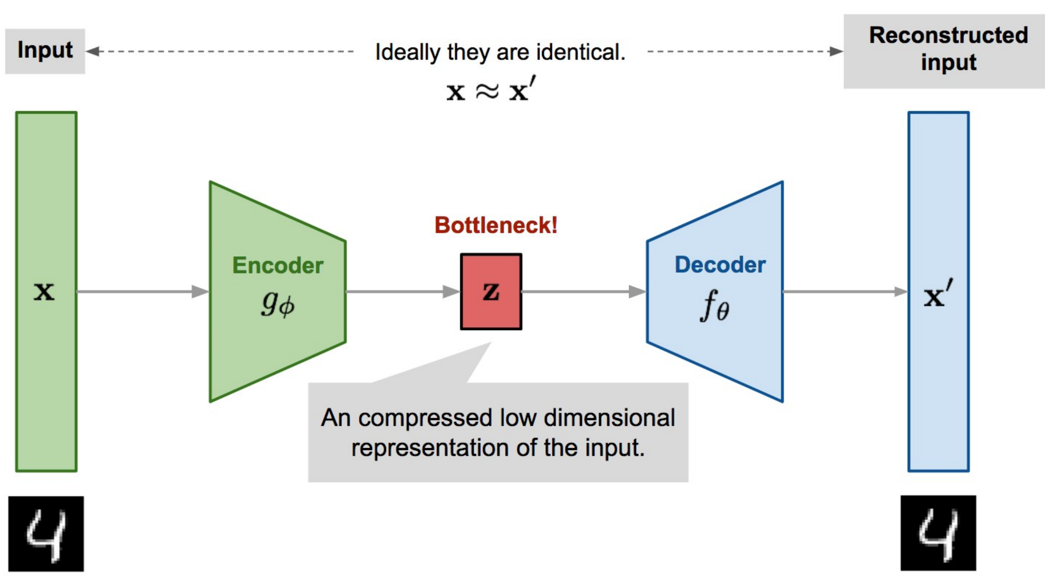


Fig. 1. Illustration of autoencoder model architecture.

The encoder network essentially accomplishes th dimensionality reductic, just like how we would use Principal Component Analysis (PCA) or Matrix Factorization (MF) for. In addition, the autoencod explicitly optimized for the data reconstruction from the code. A good intermediate representat only can capture latent variables, but also benefits a f decompressior process.

The model contains an encoder functio $g(.)$ parameterized by ϕ and a decoder functior $f(.)$ parameterized by θ. The low-dimensional code learned for inp $\mathbf{x}$ in the bottleneck layer is $\mathbf{z} =$ and the reconstructed input is $\mathbf{x}' = f_\theta(g_\phi(\mathbf{x}))$.

The parameters (θ, ϕ) are learned together to output a reconstructed data sample same as the orig input, $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$, or in other words, to learn an identity function. There are various metric quantify the difference between two vectors, such as cross entropy when the activation funct sigmoid, or as simple as MSE loss

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

Denoising Autoencoder

Since the autoencoder learns the identity function, we are facing the risk of "overfitting" when the more network parameters than the number of data po

To avoid overfitting and improve the robustne **Denoising Autoencoder** (Vincent et al. 2008) proposed a modification to the basic autoencoder. The input is partially corrupted by adding noises masking some values of the input vector in a stochastic mann $\tilde{\mathbf{x}} \sim \mathcal{M}_\mathcal{D}(\tilde{\mathbf{x}} \vert \mathbf{x})$. Then the model is trained to recover the original inpu **Note: Not the corrupt one!**).

$$\begin{aligned}
\tilde{\mathbf{x}}^{(i)} &\sim \mathcal{M}_\mathcal{D}(\tilde{\mathbf{x}}^{(i)} \vert \mathbf{x}^{(i)})\\
L_\text{DAE}(\theta, \phi) &= \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)} - f_\theta(g_\phi(\tilde{\mathbf{x}}^{(i)})))^2
\end{aligned}$$

where $\mathcal{M}_\mathcal{D}$ defines the mapping from the true data samples to the noisy or corrupted (

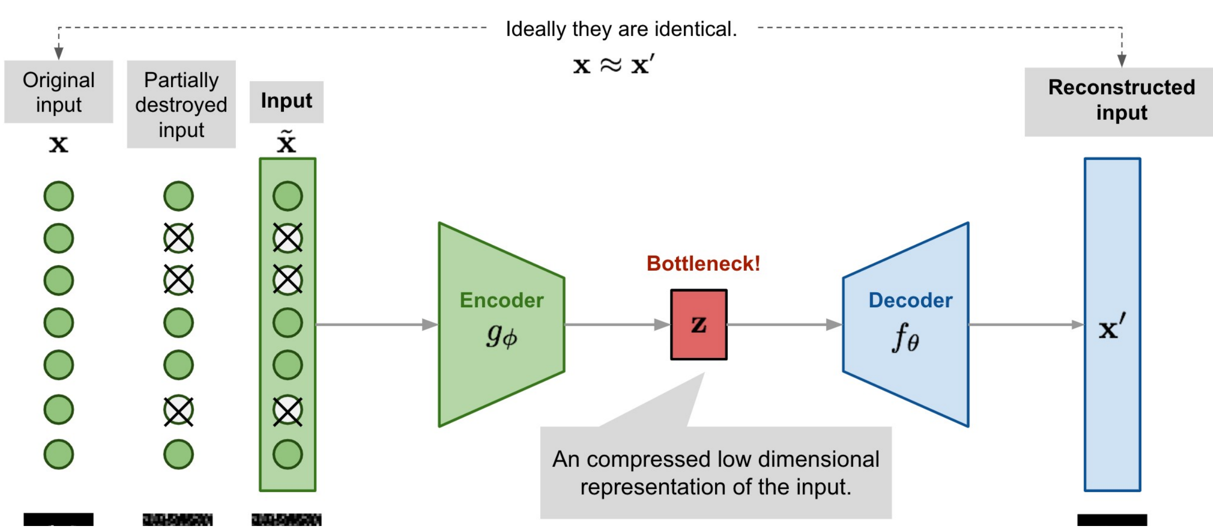


Fig. 2. Illustration of denoising autoencoder model architecture.

This design is motivated by the fact that humans can easily recognize an object or a scene even view is partially occluded or corrupted. To “repair” the partially destroyed input, the den autoencoder has to discover and capture relationship between dimensions of input in order to missing pieces.

For high dimensional input with high redundancy, like images, the model is likely to depend on evic gathered from a combination of many input dimensions to recover the denoised version (sounds li attention mechanism, right?) rather than to overfit one dimension. This builds up a good foundati learning *robust* latent representation

The noise is controlled by a stochastic mappin $\mathcal{M}_\mathcal{D}(\tilde{\mathbf{x}}|\mathbf{x})$, and it is not specific to a particular type o corruption process (i.e. masking noise, Gaussian noise, salt-and-pepper noise, etc.). Naturally corruption process can be equipped with prior knowle

In the experiment of the original DAE paper, the noise is applied in this way: a fixed proportion of dimensions are selected at random and their values are forced to 0. Sounds a lot like dropout, ri Well, the denoising autoencoder was proposed in 2008, 4 years before the dropout pa Hinton, et al. 2012) ;)

Sparse Autoencoder

Sparse Autoencoder applies a “sparse” constraint on the hidden unit activation to avoid overfitting improve robustness. It forces the model to only have a small number of hidden units being activa the same time, or in other words, one hidden neuron should be inactivate most of

Recall that common activation function include sigmoid, tanh, relu, leaky relu, etc. A neuron is activat when the value is close to 1 and inactivate with a value close to

Let’s say there are s_l neurons in the l-th hidden layer and the activation function for j-th neuron in this layer is labelled as $a^{(l)}_j(.)$, $j=1,\dots,s_l$. The fraction of activation of this neur $\hat{\rho}_j$ is expected to be a small number ρ, known as *sparsity parameter*, a common config is $\rho = 0.05$.

$$\hat{\rho}_j^{(l)} = \frac{1}{n} \sum_{i=1}^n [a_j^{(l)}(\mathbf{x}^{(i)})] \approx \rho$$

This constraint is achieved by adding a penalty term into the loss function. The KL-diverge D_KL measures the difference between two Bernoulli distributions, one with m ρ and the other with mean $\hat{\rho}_j^{(l)}$. The hyperparameter β controls how strong the penalty we want to apply on the sparsity l

$$\begin{aligned} L_{\text{SAE}}(\theta) &= L(\theta) + \beta \sum_{l=1}^{L} \sum_{j=1}^{s_l} D_{\text{KL}}(\rho \| \hat{\rho}_j^{(l)}) \\ &= L(\theta) + \beta \sum_{l=1}^{L} \sum_{j=1}^{s_l} \rho \log \frac{\rho}{\hat{\rho}_j^{(l)}} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j^{(l)}} \end{aligned}$$

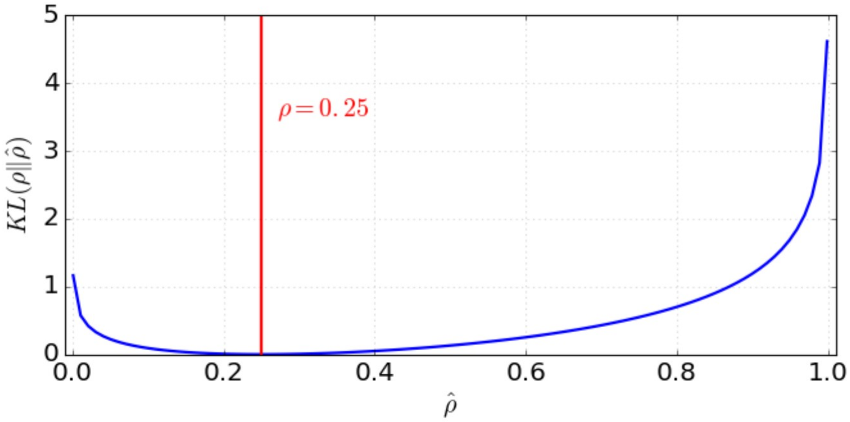
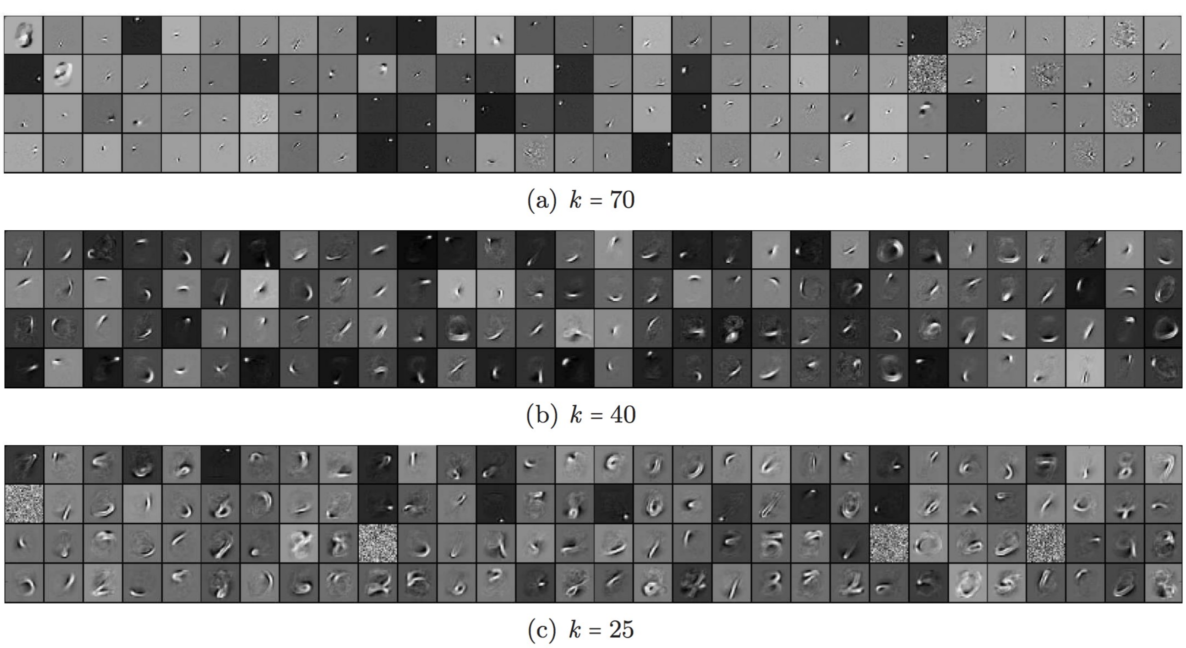
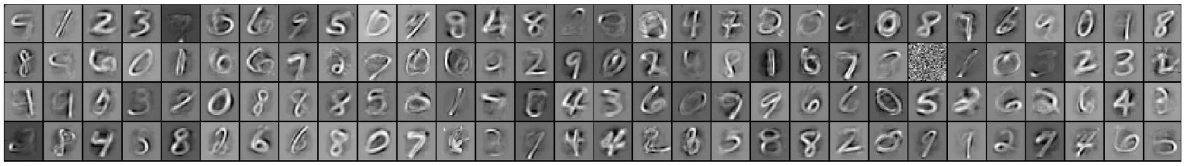


Fig. 4. The KL divergence between a Bernoulli distribution with mean $\rho = 0.25$ *and a Bernoulli distribution with mean* $0 \leq \hat{\rho} \leq 1$.

k-Sparse Autoencoder

In k-Sparse Autoencoder Makhzani and Frey, 2013), the sparsity is enforced by only keeping the top highest activations in the bottleneck layer with linear activation function. First we run feedfo through the encoder network to get the compressed co$\mathbf{z} = g(\mathbf{x})$. Sort the values in the code vecto $\mathbf{z}$. Only the k largest values are kept while other neurons are set to 0. This can be done in a ReLU la with an adjustable threshold too. Now we have a sparsified co$\mathbf{z}' = \text{Sparsify}(\mathbf{z})$. Compute the output and the loss from the sparsified cod$L = \|\mathbf{x} - f(\mathbf{z}')\|_2^2$. And, the back-propagation only goe through the top k activated hidden uni





(d) $k = 10$

Fig. 5. Filters of the k-sparse autoencoder for different sparsity levels k, learnt from MNIST with 1000 hidden units.. (Image source: Makhzani and Frey, 2013)

Contractive Autoencoder

Similar to sparse autoencode **Contractive Autoencoder** (Rifai, et al, 2011) encourages the learnec representation to stay in a contractive space for better robustn

It adds a term in the loss function to penalize the representation being too sensitive to the inpu thus improve the robustness to small perturbations around the training data points. The sensit measured by the Frobenius norm of the Jacobian matrix of the encoder activations with respect t input:

$$\|J_f(\mathbf{x})\|_F^2 = \sum_{ij} \Big(\frac{\partial h_j(\mathbf{x})}{\partial x_i} \Big)^2$$

where h_j is one unit output in the compressed co $\mathbf{z} = f(x)$.

This penalty term is the sum of squares of all partial derivatives of the learned encoding with resp input dimensions. The authors claimed that empirically this penalty was found to carve a represen that corresponds to a lower-dimensional non-linear manifold, while staying more invariant to m directions orthogonal to the manifc

VAE: Variational Autoencoder

The idea of **Variational Autoencoder** (Kingma & Welling, 2014), short for **VAE**, is actually less similar to all the autoencoder models above, but deeply rooted in the methods of variational bayesiar graphical model

Instead of mapping the input int *fixed* vector, we want to map it into a distribution. Let's label t distribution a p_θ, parameterized by θ. The relationship between the data inp $\mathbf{x}$ and the latent encoding vecto $\mathbf{z}$ can be fully defined by

- Prior $p_\theta(\mathbf{z})$
- Likelihood $p_\theta(\mathbf{x}|\mathbf{z})$
- Posterior $p_\theta(\mathbf{z}|\mathbf{x})$

Assuming that we know the real paramet θ^* for this distribution. In order to generate a sample t looks like a real data poin$\mathbf{x}^{(i)}$, we follow these steps

1. First, sample a $\mathbf{z}^{(i)}$ from a prior distributio$p_{\theta^*}(\mathbf{z})$.
2. Then a value $\mathbf{x}^{(i)}$ is generated from a conditional distributi$p_{\theta^*}(\mathbf{x}|\mathbf{z} = \mathbf{z}^{(i)})$.

The optimal parameteθ^* is the one that maximizes the probability of generating real data sam|

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

Commonly we use the log probabilities to convert the product on RHS to a s

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

Now let's update the equation to better demonstrate the data generation process so as to invol encoding vector

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

Unfortunately it is not easy to compu$p_\theta(\mathbf{x}^{(i)})$ in this way, as it is very expensive to check all th possible values of $\mathbf{z}$ and sum them up. To narrow down the value space to facilitate faster search, would like to introduce a new approximation function to output what is a likely code given an $\mathbf{x}$, $q_\phi(\mathbf{z}|\mathbf{x})$, parameterized by ϕ.

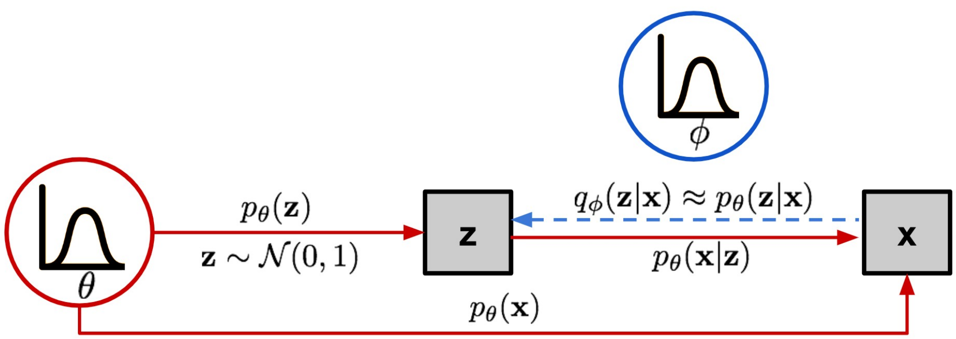


Fig. 6. The graphical model involved in Variational Autoencoder. Solid lines denote the generative distribution $p_\theta(.)$ and dashed lines denote the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

Now the structure looks a lot like an autoencod

- The conditional probabilit$p_\theta(\mathbf{x}|\mathbf{z})$ defines a generative model, similar to the decod$f_\theta(\mathbf{x}|\mathbf{z})$ introduced above $p_\theta(\mathbf{x}|\mathbf{z})$ is also known as *probabilistic decoder.*
- The approximation functio$q_\phi(\mathbf{z}|\mathbf{x})$ is the *probabilistic encoder*, playing a similar role as $g_\phi(\mathbf{z}|\mathbf{x})$ above.

Loss Function: ELBO

The estimated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ should be very close to the real one $p_\theta(\mathbf{z}|\mathbf{x})$. We can use Kullback-Leibler divergence to quantify the distance between these two distributions. KL divergence $D_\text{KL}(X\|Y)$ measures how much information is lost if the distribution Y is used to represe

In our case we want to minimize $D_\text{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$ with respect to ϕ.

But why use $D_\text{KL}(q_\phi\|p_\theta)$ (reversed KL) instead of $D_\text{KL}(p_\theta\|q_\phi)$ (forward KL)? Eric Jang has a great explanation in his post on Bayesian Variational methods. As a quick reca

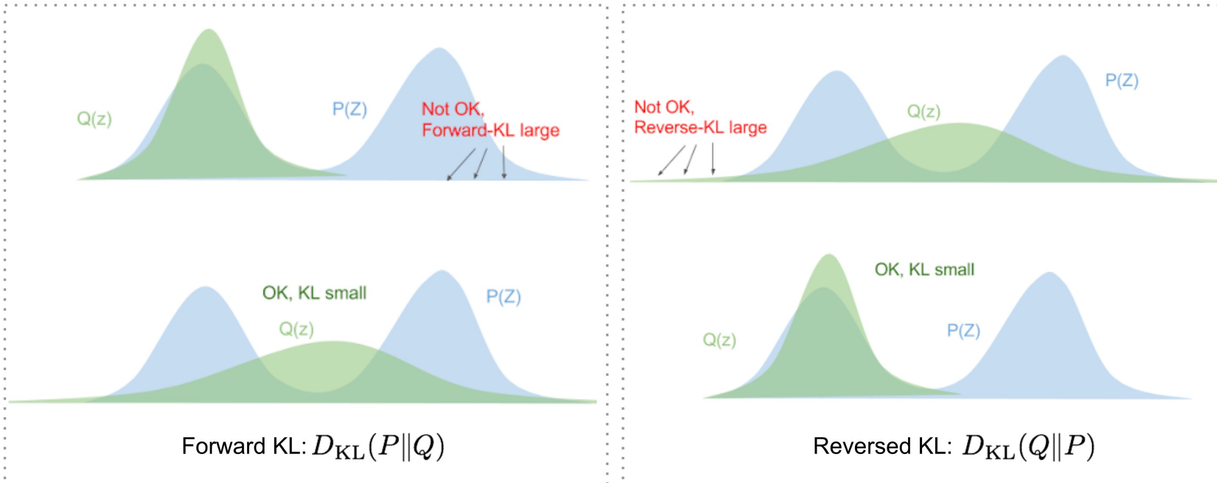


Fig. 7. Forward and reversed KL divergence have different demands on how to match two distributions. (Image source: blog.evjang.com/2016/08/variational-bayes.html)

- Forward KL divergence: $D_\text{KL}(P\|Q) = \mathbb{E}_{z\sim P(z)} \log\frac{P(z)}{Q(z)}$; we have to ensure that Q(z)>0 wherever P(z)>0. The optimized variational distribution $q(z)$ has to cover over the entire $p(z)$.
- Reversed KL divergence: $D_\text{KL}(Q\|P) = \mathbb{E}_{z\sim Q(z)} \log\frac{Q(z)}{P(z)}$; minimizing the reversed KL divergence squeezes the $Q(z)$ under $P(z)$.

Let's now expand the equatio

$$
\begin{aligned}
& D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \\
& = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
& = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} & \text{; Because } p(z|x) = p(z,x)/p(x) \\
& = \int q_\phi(\mathbf{z}|\mathbf{x}) \big(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \big) d\mathbf{z} \\
& = \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} & \text{; Because } \int q(z|x) dz = 1 \\
& = \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} & \text{; Because } p(z,x) = p(x|z)p(z) \\
& = \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})] \\
& = \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})
\end{aligned}
$$

So we have:

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

Once rearrange the left and right hand side of the equat

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

The LHS of the equation is exactly what we want to maximize when learning the true distribution want to maximize the (log-)likelihood of generating real data (th$\log p_\theta(\mathbf{x})$) and also minimize the difference between the real and estimated posterior distributions (the D_{KL} works like a regularizer). Note tha$p_\theta(\mathbf{x})$ is fixed with respect tq_ϕ.

The negation of the above defines our loss functi

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

In Variational Bayesian methods, this loss function is known as t*variational lower boung* or *evidence lower bound*. The “lower bound” part in the name comes from the fact that KL divergence is always negative and thus$-L_{\mathrm{VAE}}$ is the lower bound o$\log p_\theta(\mathbf{x})$.

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

Therefore by minimizing the loss, we are maximizing the lower bound of the probability of gene real data samples

Reparameterization Trick

The expectation term in the loss function invokes generating samples f$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Sampling is a stochastic process and therefore we cannot backpropagate the gradient. To make it trainable reparameterization trick is introduced: It is often possible to express the random va$\mathbf{z}$ as a deterministic variabl$\mathbf{z} = \mathcal{T}_\phi(\mathbf{x}, \boldsymbol{\epsilon})$, where $\boldsymbol{\epsilon}$ is an auxiliary independent random variable, and t transformation functio$\mathcal{T}_\phi$ parameterized by ϕ converts $\boldsymbol{\epsilon}$ to $\mathbf{z}$.

For example, a common choice of the form $q_\phi(\mathbf{z}|\mathbf{x})$ is a multivariate Gaussian with a diagona covariance structure

$$
\begin{aligned}
&\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) & \\
&\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}) & \scriptstyle{\text{; Reparameterization trick.}}
\end{aligned}
$$

where $\odot$ refers to element-wise produc

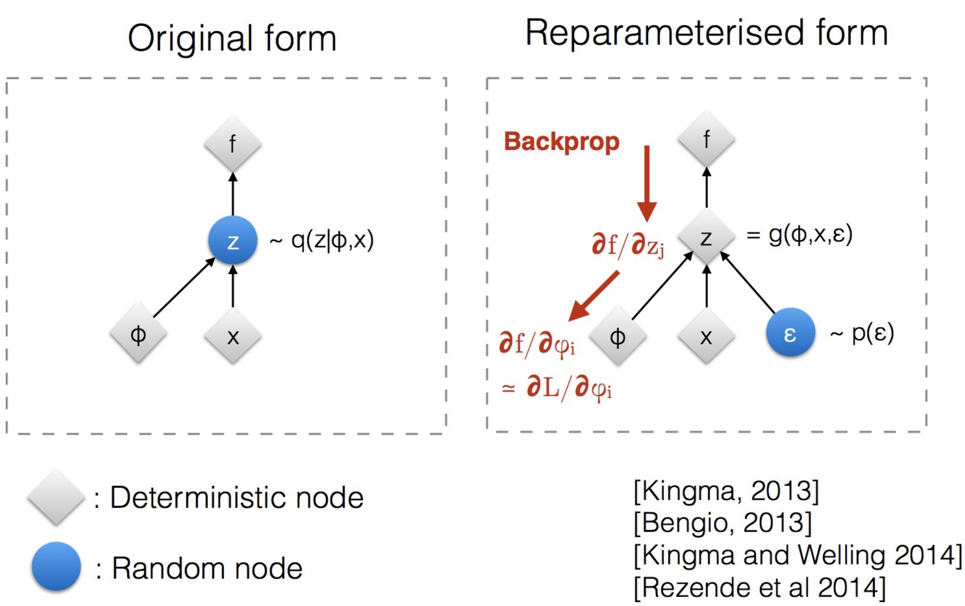
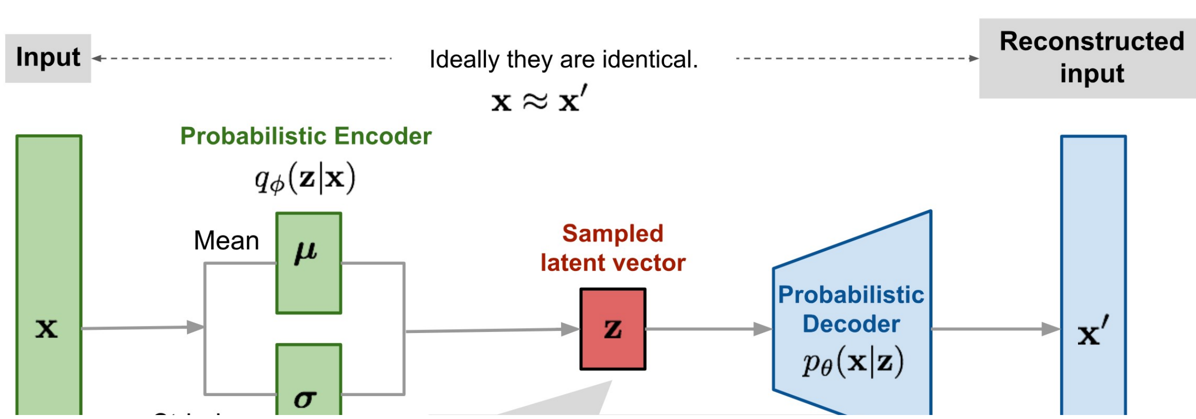


Fig. 8. Illustration of how the reparameterization trick makes the $\mathbf{z}$ sampling process trainable.(Image source: Slide 12 in Kingma's NIPS 2015 workshop talk)

The reparameterization trick works for other types of distributions too, not only Gaussian. I multivariate Gaussian case, we make the model trainable by learning the mean and variance o distribution μ and σ, explicitly using the reparameterization trick, while the stochasticity remains ir random variable $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$.



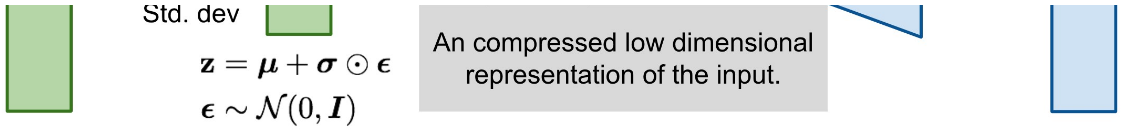


Fig. 9. Illustration of variational autoencoder model with the multivariate Gaussian assumptior

Beta-VAE

If each variable in the inferred latent representati$\mathbf{z}$ is only sensitive to one single generative facto and relatively invariant to other factors, we will say this representation is disentangled or factorized benefit that often comes with disentangled representatio*good interpretability*and easy generalization to a variety of tasks

For example, a model trained on photos of human faces might capture the gentle, skin color, hair hair length, emotion, whether wearing a pair of glasses and many other relatively independent fact separate dimensions. Such a disentangled representation is very beneficial to facial image genera

β-VAE (Higgins et al., 2017) is a modification of Variational Autoencoder with a special emphasi discover disentangled latent factors. Following the same incentive in VAE, we want to maximize probability of generating real data, while keeping the distance between the real and estimated pos distributions small (say, under a small constaδ):

$$\begin{aligned}
&\max_{\phi,\theta} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})] \\
&\text{subject to } D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) < \delta
\end{aligned}$$

We can rewrite it as a Lagrangian with a Lagrangian multiplβ under theKKT condition The above optimization problem with only one inequality constraint is equivalent to maximizing the fol equation$\mathcal{F}(\theta, \phi, \beta)$:

$$\begin{aligned}
\mathcal{F}(\theta, \phi, \beta) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \beta(D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \delta) \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) + \beta\delta \\
&\geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) & \text{; Because } \beta,\delta\geq 0
\end{aligned}$$

The loss function ofβ-VAE is defined as:

$$L_{\text{BETA}}(\phi, \beta) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + \beta D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))$$

where the Lagrangian multipliβ is considered as a hyperparamete

Since the negation of$L_{\text{BETA}}(\phi, \beta)$ is the lower bound of the Lagrangia$\mathcal{F}(\theta, \phi, \beta)$. Minimizing the loss is equivalent to maximizing the Lagrangian and thus works for our initial optimization pr

When $\beta = 1$, it is same as VAE. When $\beta > 1$, it applies a stronger constraint on the latent bottlene and limits the representation capacity $\mathbf{z}$. For some conditionally independent generative facto keeping them disentangled is the most efficient representation. Therefore a hiβ encourages more efficient latent encoding and further encourages the disentanglement. Meanwhile, a hβ may create a trade-off between reconstruction quality and the extent of disentangler

Burgess, et al. (2017) discussed the distentangling iβ-VAE in depth with an inspiration by th information bottleneck thec and further proposed a modification β-VAE to better control the encoding representation capacit

VQ-VAE and VQ-VAE-2

The **VQ-VAE** ("Vector Quantised-Variational AutoEncode van den Oord, et al. 2017) model learns a discrete latent variable by the encoder, since discrete representations may be a more natural f problems like language, speech, reasoning, et

Vector quantisation (VQ) is a method to ma K-dimensional vectors into a finite set of "code" vectc The process is very much similar t KNN algorithm. The optimal centroid code vector that a sam should be mapped to is the one with minimum euclidean dista

Let $\mathbf{e} \in \mathbb{R}^{K \times D}, i = 1, \dots, K$ be the latent embedding space (also known as "codebook") in VQ-V where K is the number of latent variable categories a D is the embedding size. An individu embedding vector i $\mathbf{e}_i \in \mathbb{R}^D, i = 1, \dots, K$.

The encoder outpu $E(\mathbf{x}) = \mathbf{z}_e$ goes through a nearest-neighbor lookup to match to on K embedding vectors and then this matched code vector becomes the input for the de $D(.)$:

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

Note that the discrete latent variables can have different shapes in differnet applications; for exa 1D for speech, 2D for image and 3D for vide

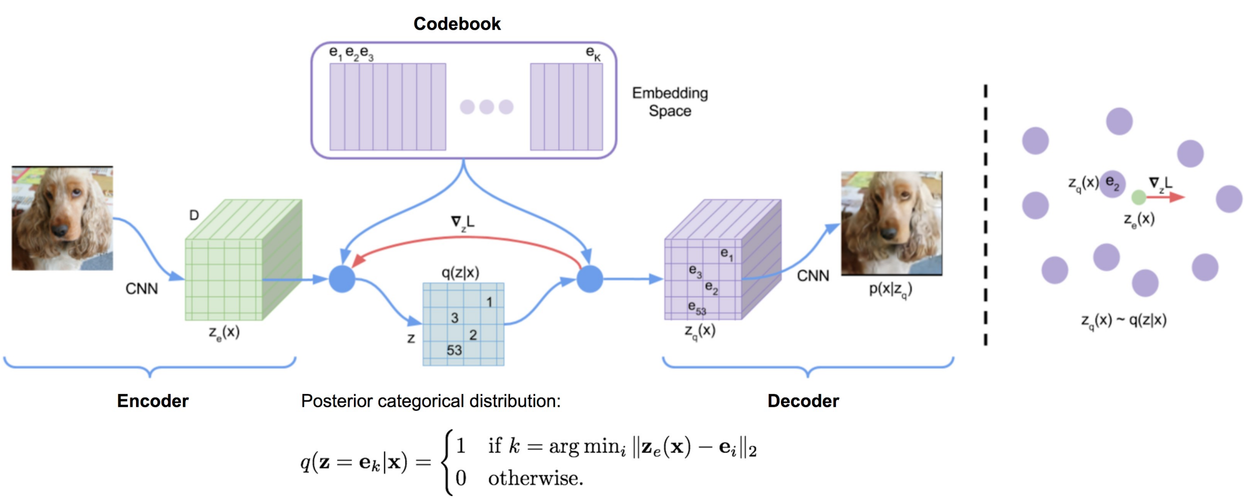


Fig. 10. The architecture of VQ-VAE (Image source: van den Oord, et al. 2017)

Because argmin() is non-differentiable on a discrete space, the gradie $\nabla_z L$ from decoder inpu $\mathbf{z}_q$ is copied to the encoder outp $\mathbf{z}_e$. Other than reconstruction loss, VQ-VAE also optimize

- *VQ loss* : The L2 error between the embedding space and the encoder outp
- *Commitment loss*: A measure to encourage the encoder output to stay close to the embedc space and to prevent it from fluctuating too frequently from one code vector to an

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

where$\text{sq}[.]$ is the `stop_gradient` operator

The embedding vectors in the codebook is updated through EMA (exponential moving average). Giv code vector$\mathbf{e}_i$, say we have n_i encoder output vectors$\{\mathbf{z}_{i,j}\}_{j=1}^{n_i}$, that are quantized to$\mathbf{e}_i$:

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

where (t) refers to batch sequence in timeN_i and $\mathbf{m}_i$ are accumulated vector count and volum respectively.

VQ-VAE-2 (Ali Razavi, et al. 2019) is a two-level hierarchical VQ-VAE combined with self-attentic autoregressive mode

1. Stage 1 is to **train a hierarchical VQ-VAE**: The design of hierarchical latent variables intends separate local patterns (i.e., texture) from global information (i.e., object shapes). The training c larger bottom level codebook is conditioned on the smaller top level code too, so that it doe have to learn everything from scratc
2. Stage 2 is to **learn a prior over the latent discrete codebook** so that we sample from it anc generate images. In this way, the decoder can receive input vectors sampled from a sir distribution as the one in training. A powerful autoregressive model enhanced with multi-head attention layers is used to capture the prior distribution (PixelSNAIL; Chen et al 2017).

Considering that VQ-VAE-2 depends on discrete latent variables configured in a simple hierarch setting, the quality of its generated images are pretty amaz

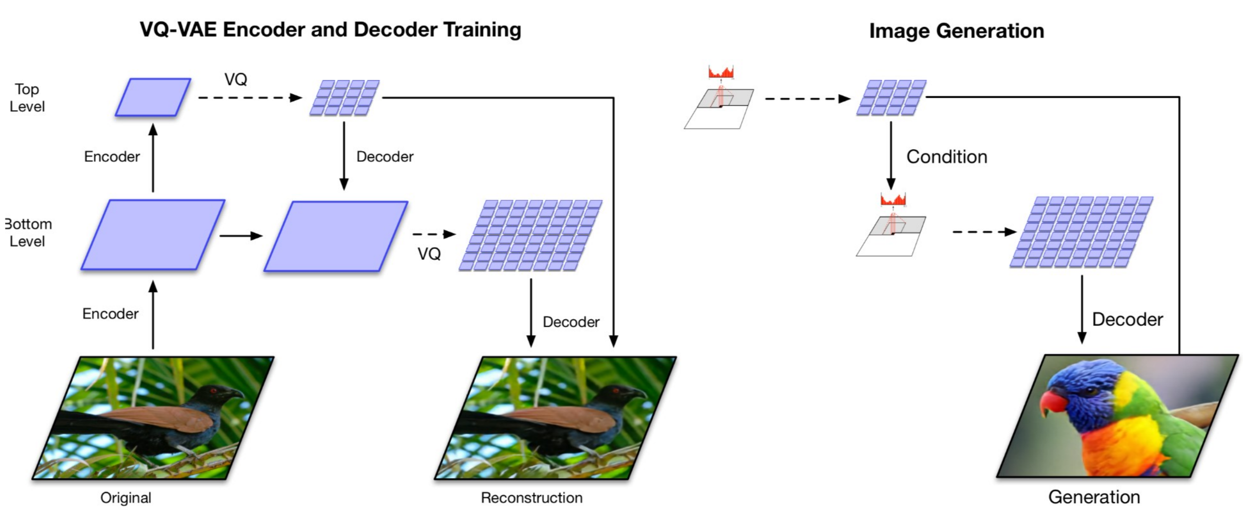


Fig. 11. Architecture of hierarchical VQ-VAE and multi-stage image generation. (Image source: Ali Razavi, et al. 2019)

Algorithm 1 VQ-VAE training (stage 1)

Require: Functions E_{top} , E_{bottom}, D, $\mathbf{x}$ (batch of training images)

$\mathbf{h}_{top} \leftarrow E_{top}(\mathbf{x})$

▷ quantize with top codebook eq 1

$\mathbf{e}_{top} \leftarrow Quantize(\mathbf{h}_{top})$

$\mathbf{h}_{bottom} \leftarrow E_{bottom}(\mathbf{x}, \mathbf{e}_{top})$

▷ quantize with bottom codebook eq 1

$\mathbf{e}_{bottom} \leftarrow Quantize(\mathbf{h}_{bottom})$

$\hat{\mathbf{x}} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$

▷ Loss according to eq 2

$\theta \leftarrow Update(\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}))$

Algorithm 2 Prior training (stage 2)

$\mathbf{T}_{top}, \mathbf{T}_{bottom} \leftarrow \emptyset$ ▷ training set

for $\mathbf{x} \in$ training set **do**

$\quad \mathbf{e}_{top} \leftarrow Quantize(E_{top}(\mathbf{x}))$

$\quad \mathbf{e}_{bottom} \leftarrow Quantize(E_{bottom}(\mathbf{x}, \mathbf{e}_{top}))$

$\quad \mathbf{T}_{top} \leftarrow \mathbf{T}_{top} \cup \mathbf{e}_{top}$

$\quad \mathbf{T}_{bottom} \leftarrow \mathbf{T}_{bottom} \cup \mathbf{e}_{bottom}$

end for

$p_{top} = \texttt{TrainPixelCNN}(\mathbf{T}_{top})$

$p_{bottom} = \texttt{TrainCondPixelCNN}(\mathbf{T}_{bottom}, \mathbf{T}_{top})$

▷ Sampling procedure

while true **do**

$\quad \mathbf{e}_{top} \sim p_{top}$

$\quad \mathbf{e}_{bottom} \sim p_{bottom}(\mathbf{e}_{top})$

$\quad \mathbf{x} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$

end while

Fig. 12. The VQ-VAE-2 algorithm. (Image source: Ali Razavi, et al. 2019)

TD-VAE

TD-VAE (“Temporal Difference VAE” Gregor et al., 2019) works with sequential data. It relies on thr main ideas, described below

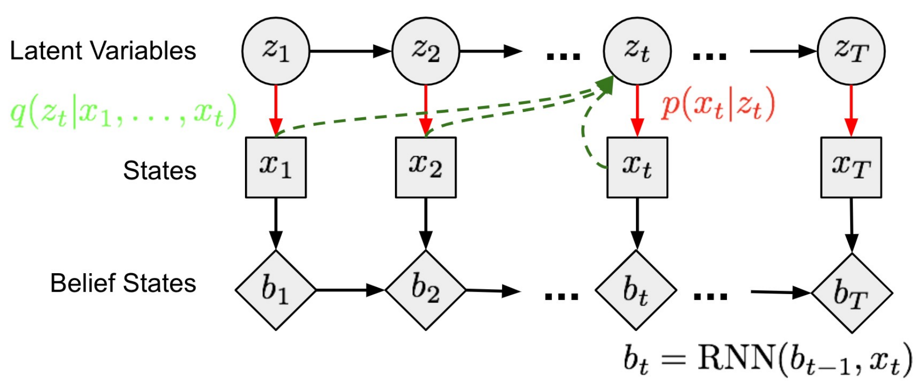


Fig. 13. State-space model as a Markov Chain model.

1. State-Space Models

In (latent) state-space models, a sequence of unobserved hidden sta$\mathbf{z} = (z_1, \dots, z_T)$ determine the observation state$\mathbf{x} = (x_1, \dots, x_T)$. Each time step in the Markov chain model in Fig. 13 can b trained in a similar manner as in Fig. 6, where the intractable poste$p(z|x)$ is approximated by a function$q(z|x)$.

2. Belief State

An agent should learn to encode all the past states to reason about the future, name*belief state*, $b_t = belief(x_1, \dots, x_t) = belief(b_{t-1}, x_t)$. Given this, the distribution of future states conditior on the past can be written as $p(x_{t+1}, \dots, x_T \vert x_1, \dots, x_t) \approx p(x_{t+1}, \dots, x_T \vert b_t)$. The hidden states in a recurrent policy are used as the agent's belief state in TD-VAE. Thus we have $b_t = \text{RNN}(b_{t-1}, x_t)$.

3. Jumpy Prediction

Further, an agent is expected to imagine distant futures based on all the information gathered so far, suggesting the capability of making jumpy predictions, that is, predicting states several steps further into the future.

Recall what we have learned from the variance lower bound above:

$$\begin{aligned}
\log p(x) &\geq \log p(x) - D_\text{KL}(q(z|x)\|p(z|x)) \\
&= \mathbb{E}_{z\sim q}\log p(x|z) - D_\text{KL}(q(z|x)\|p(z)) \\
&= \mathbb{E}_{z \sim q} \log p(x|z) - \mathbb{E}_{z \sim q} \log\frac{q(z|x)}{p(z)} \\
&= \mathbb{E}_{z \sim q}[\log p(x|z) -\log q(z|x) + \log p(z)] \\
&= \mathbb{E}_{z \sim q}[\log p(x, z) -\log q(z|x)] \\
\log p(x) &\geq \mathbb{E}_{z \sim q}[\log p(x, z) -\log q(z|x)]
\end{aligned}$$

Now let's model the distribution of the state x_t as a probability function conditioned on all the past states $x_{<t}$ and two latent variables, z_t and z_{t-1}, at current time step and one step back:

$$\log p(x_t|x_{<t}) \geq \mathbb{E}_{(z_{t-1}, z_t) \sim q}[\log p(x_t, z_{t-1}, z_{t}|x_{<t}) -\log q(z_{t-1}, z_t|x_{\leq t})]$$

Continue expanding the equation:

$$\begin{aligned}
& \log p(x_t|x_{<t}) \\
&\geq \mathbb{E}_{(z_{t-1}, z_t) \sim q}[\log p(x_t, z_{t-1}, z_{t}|x_{<t}) -\log q(z_{t-1}, z_t|x_{\leq t})] \\
&\geq \mathbb{E}_{(z_{t-1}, z_t) \sim q}[\log p(x_t|\color{red}{z_{t-1}}, z_{t}, \color{red}{x_{<t}}) + \color{blue}{\log p(z_{t-1}, z_{t}|x_{<t})} -\log q(z_{t-1}, z_t|x_{\leq t})] \\
&\geq \mathbb{E}_{(z_{t-1}, z_t) \sim q}[\log p(x_t|z_{t}) + \color{blue}{\log p(z_{t-1}|x_{<t})} + \color{blue}{\log p(z_{t}|z_{t-1})} - \color{green}{\log q(z_{t-1}, z_t|x_{\leq t})}] \\
&\geq \mathbb{E}_{(z_{t-1}, z_t) \sim q}[\log p(x_t|z_{t}) + \log p(z_{t-1}|x_{<t}) + \log p(z_{t}|z_{t-1}) - \color{green}{\log q(z_t|x_{\leq t})} - \color{green}{\log q(z_{t-1}|z_t, x_{\leq t})}]
\end{aligned}$$

Notice two things:

- The red terms can be ignored according to Markov assumptions.
- The blue term is expanded according to Markov assumptions.
- The green term is expanded to include an one-step prediction back to the past as a smoothing distribution.

Precisely, there are four types of distributions to learn:

1. $p_D(.)$ is the **decoder** distribution:
   - $p(x_t \mid z_t)$ is the encoder by the common definition;
   - $p(x_t \mid z_t) \to p_D(x_t \mid z_t)$;
2. $p_T(.)$ is the **transition** distribution:
   - $p(z_t \mid z_{t-1})$ captures the sequential dependency between latent variables;
   - $p(z_t \mid z_{t-1}) \to p_T(z_t \mid z_{t-1})$;

3. $p_B(.)$ is the **belief** distribution:
   - Both $p(z_{t-1} \mid x_{<t})$ and $q(z_t \mid x_{\leq t})$ can use the belief states to predict the latent variabl
   - $p(z_{t-1} \mid x_{<t}) \to p_B(z_{t-1} \mid b_{t-1})$;
   - $q(z_t \mid x_{\leq t}) \to p_B(z_t \mid b_t)$;
4. $p_S(.)$ is the **smoothing** distribution:
   - The back-to-past smoothing term $q(z_{t-1} \mid z_t, x_{\leq t})$ can be rewritten to be dependent of beli states too;
   - $q(z_{t-1} \mid z_t, x_{\leq t}) \to p_S(z_{t-1} \mid z_t, b_{t-1}, b_t)$;

To incorporate the idea of jumpy prediction, the sequential ELBO has to not only work on $t, t+1$, but also two distant timestamps $t_1 < t_2$. Here is the final TD-VAE objective function to maximiz

$$J_{t_1,t_2} = \mathbb{E}[\log p_D(x_{t_2}|z_{t_2}) + \log p_B(z_{t_1}|b_{t_1}) + \log p_T(z_{t_2}|z_{t_1}) - \log p_B(z_{t_2}|b_{t_2}) - \log p_S(z_{t_1}|z_{t_2}, b_{t_1}, b_{t_2})]$$

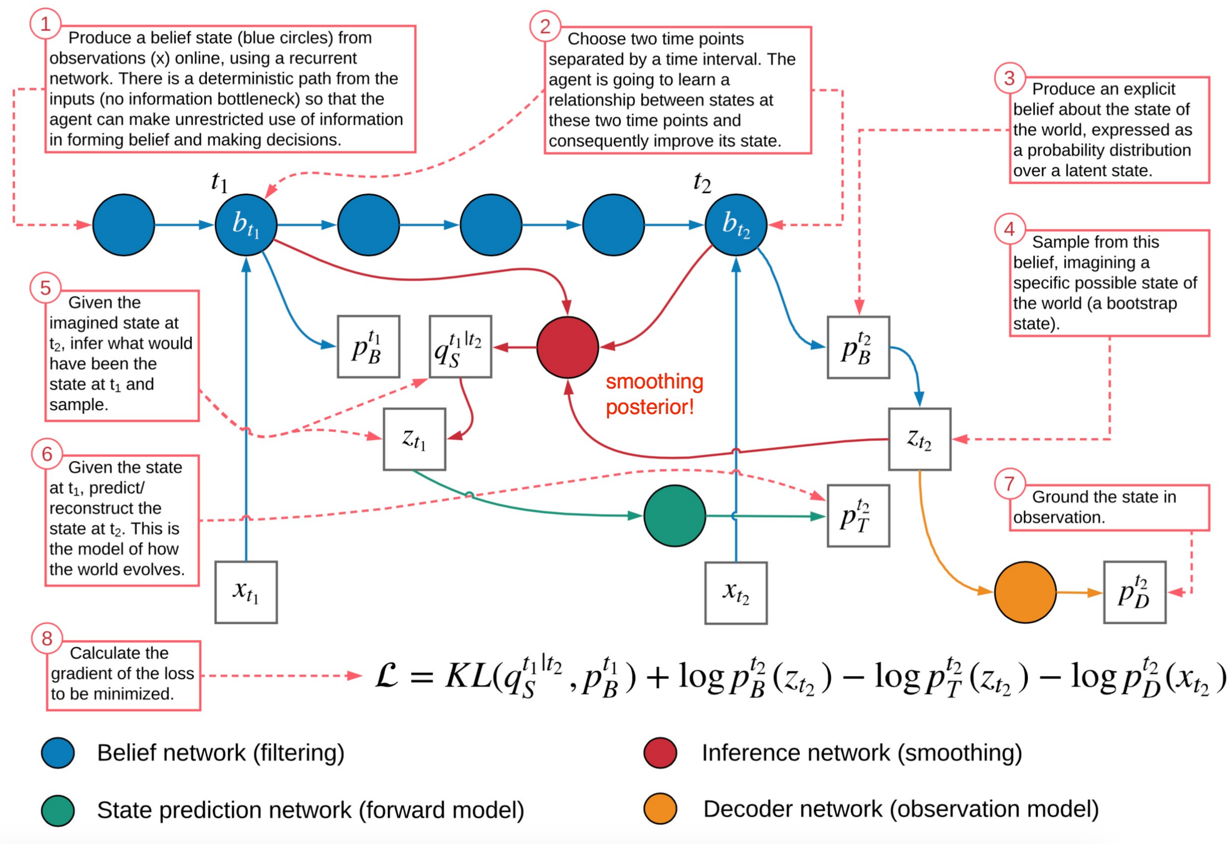


Fig. 14. A detailed overview of TD-VAE architecture, very nicely done. (Image source: TD-VAE paper)

Cited as:

```
@article{weng2018VAE,
  title   = "From Autoencoder to Beta-VAE",
  author  = "Weng, Lilian",
  journal = "lilianweng.github.io/lil-log",
  year    = "2018",
```

```
  url      = "http://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-be
}
```

References

[1] Geoffrey E. Hinton, and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." Science 313.5786 (2006): 504-507.

[2] Pascal Vincent, et al. "Extracting and composing robust features with denoising autoencoders." ICML, 2008.

[3] Pascal Vincent, et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." Journal of machine learning research 11.Dec (2010): 3371-3408.

[4] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).

[5] Sparse Autoencoder by Andrew Ng

[6] Alireza Makhzani, Brendan Frey (2013). "k-sparse autoencoder". ICLR 2014.

[7] Salah Rifai, et al. "Contractive auto-encoders: Explicit invariance during feature extraction." ICML, 2011.

[8] Diederik P. Kingma, and Max Welling. "Auto-encoding variational bayes." ICLR 2014.

[9] Tutorial - What is a variational autoencoder? on jaan.io

[10] Youtube tutorial: Variational Autoencoders by Arxiv Insights

[11] "A Beginner's Guide to Variational Methods: Mean-Field Approximation" by Eric Jang.

[12] Carl Doersch. "Tutorial on variational autoencoders." arXiv:1606.05908, 2016.

[13] Irina Higgins, et al. "β-VAE: Learning basic visual concepts with a constrained variational framework." ICLR 2017.

[14] Christopher P. Burgess, et al. "Understanding disentangling in beta-VAE." NIPS 2017.

[15] Aaron van den Oord, et al. "Neural Discrete Representation Learning" NIPS 2017.

[16] Ali Razavi, et al. "Generating Diverse High-Fidelity Images with VQ-VAE-2". arXiv preprint arXiv:1906.00446 (2019).

[17] Xi Chen, et al. "PixelSNAIL: An Improved Autoregressive Generative Model." arXiv preprint arXiv:1712.09763 (2017).

[18] Karol Gregor, et al. "Temporal Difference Variational Auto-Encode ICLR 2019.

← Attention? Attention Flow-based Deep Generative Models →

10 Comments **lilianweng.github.io/lil-log** 1 **Login**

Recommend 12 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Seyyed Hossein Hasanpour • 18 days ago

Concerning the reparameterization trick and why we do that, you need to know that

the issue is not that we cannot backprop through a “random node” in any technical sense. Rather, backproping would not compute an estimate of the derivative. Without the reparameterization trick, we have no guarantee that sampling large numbers of z will help converge to the right estimate of ∇θ.
you'll find this link interesting : http://gregorygundersen.com...

• Reply • Share ›

alex • a month ago

Probably a trivial question. N(mean=m, var=v) is a a scalar drawn according to a gaussian distribution with mean equal to m and var equal to v. Of course m can be a vector of dimension d and v a matrix dXd to generate d data. What does it mean to say N(x; m, v) ? for instance under Reparameterization Trick section. Thanks for the help and cheers for your blog all !

• Reply • Share ›

Pony CHEN • 4 months ago • edited

Thanks for your post, could you tell me how to draw Figure 1 in that style, is it with the aid of Visio or some other softwares like that ?

• Reply • Share ›

Nitish Bhardwaj • 6 months ago • edited

Great post. Suggesting a small correction in "*Fig. 2. Illustration of denoising autoencoder model architecture*.", the input is **not x**, it's **noisy x, ~x**.

• Reply • Share ›

Lilian Weng Mod → Nitish Bhardwaj • 6 months ago

Fixed :)

1 • Reply • Share ›

Illiou • 7 months ago

In the first paragraph of Autoencoder the year should be 2006 instead of 2016. :)

• Reply • Share ›

GH • 10 months ago

Thanks for the post. I think there's a typo in Figure 6. The bottom solid arrow should be p_\theta(x) instead of p_\theta(z).

• Reply • Share ›

So Hyun • 10 months ago