Autoencoders & Variational Autoencoders

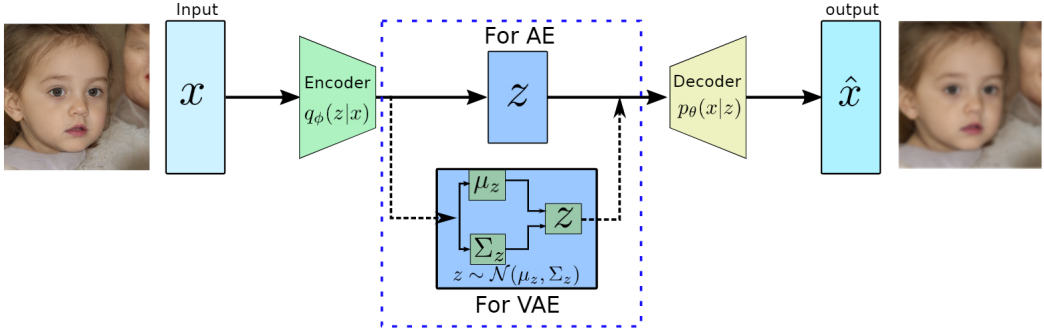
Naeemullah Khan
naeemullah.khan@kaust.edu.sa



جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

August 9, 2026



Source: data-science-blog.com, "variational-auto-encoder-image-encoding-decoding" (2022).

Table of Contents

1. Autoencoders (AE)
    1. Motivation
    2. Learning Outcomes
    3. Introduction
    4. Architecture
    5. As Generative Models
    6. Applications
    7. Challenges
2. Variational Autoencoders (VAE)
    1. Motivation
    2. Learning Outcomes

Table of Contents (cont.)

3. Introduction

4. Latent Variable Models

5. Variational Inference

6. Evidence Lower Bound (ELBO)

7. Reparameterization Trick

8. Loss Function

9. KL Divergence

10. Variants and Extensions

11. Results

12. Applications

13. Perceptual Loss

Table of Contents (cont.)

14. From VAEs to VQ-VAEs

15. VQ-VAE

16. Limitations and Challenges

3. References

Autoencoders (AE)

Autoencoders: **Motivation**

Motivation

- Real-world data is often **large**, noisy, and difficult to interpret.
- We aim to **compress** data efficiently while preserving essential information.
- It is also valuable for machines to **generate** new, realistic data samples.
- **Autoencoders** are a type of **unsupervised learning** model used for compression and representation learning; the desire to also *generate* new data motivates VAEs later in this lecture.
- They **learn** to represent data in a **lower-dimensional space**, capturing its most important features.

Autoencoders: **Learning Outcomes**

⇒ Learning Outcomes

By the end of this session, you will be able to:

- Explain the basic concept and purpose of autoencoders
- Identify and describe the main components: **encoder**, **bottleneck**, and **decoder**
- Explain why plain autoencoders fall short as generative models
- Recognize practical applications and limitations of autoencoders

Autoencoders: **Introduction**

Autoencoders

Autoencoders are a type of artificial neural network used for learning efficient codings of input data in an unsupervised manner. The goal is to learn a compressed (encoded) representation of the input and then reconstruct the original input from this compressed version.

- **Developed originally in the 1980s**, autoencoders have gained renewed popularity due to advances in deep learning and hardware.
- Often used in **dimensionality reduction, denoising, and anomaly detection**; their generative extension, the VAE, comes later in this lecture.

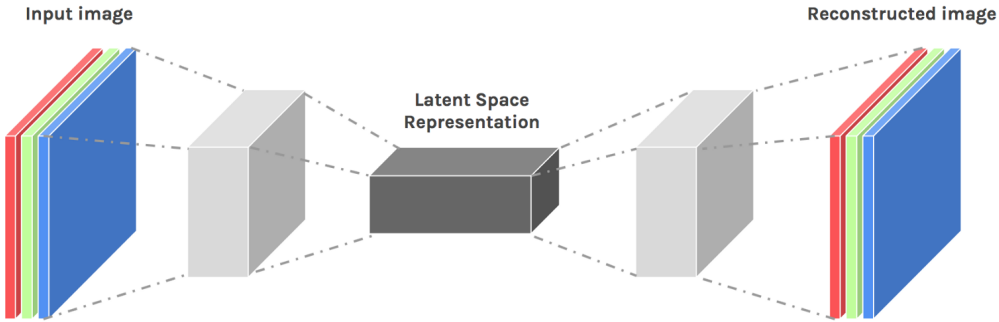
Autoencoders (cont.)

Autoencoders consist of two main components:

- **Encoder**: Maps the input data to a latent space Z, compressing the input into a lower-dimensional representation.
- **Decoder**: Reconstructs the input data from the encoded latent representation.

Variants may feed altered or corrupted versions of the input to the network while keeping the clean input as the reconstruction target, which can help the model learn more robust features (e.g., denoising autoencoders).

Autoencoders (cont.)



Autoencoder architecture

Source: V. Jayawardana, "Autoencoders — Bits and Bytes of Deep Learning", Towards Data Science, 2017.

Autoencoders (cont.)

- **Bottleneck (Latent Space)**
  - The bottleneck is a small hidden layer in the middle of the network.
  - It forces the network to squeeze the input into a shorter, simpler code.
  - This helps the network find and keep only the most important information.
  - A small bottleneck stops the autoencoder from just memorizing the input.
- **Loss Function**
  - Tells us how close the output is to the original input.
  - The most common loss is **Mean Squared Error (MSE)**: it measures the average squared difference between input and output.
  - For special cases, we can use other losses, like binary cross-entropy for yes/no data, or KL divergence for special types of autoencoders.
  - The loss function guides the network to learn better and more useful features.

Autoencoders: **Architecture**

Autoencoders: Architecture

Design Considerations:

- **How deep?** More layers can learn more, but also make training harder and risk overfitting.
- **How wide?** More neurons per layer can help, but too many may just copy the input instead of learning.
- **Symmetry:** Encoder and decoder can look the same, but don't have to; sometimes different shapes work better.
- **Regularization:** Tricks like dropout or weight penalties help prevent the model from just memorizing.
- **Latent size:** The "bottleneck" size is key: too small loses info, too big doesn't compress.
- **Activation:** Functions like ReLU or sigmoid help the network learn complex patterns.

Autoencoders: Architecture (cont.)

Encoder:

- Takes input data (e.g., image, text, audio)
- Uses neural layers to reduce it to a smaller dimension
- Learns to keep only the important features
- *Example*: 784-pixel image to 32 numbers

Autoencoders: Architecture (cont.)

Latent Space (Bottleneck):

- This is the core representation of the data
- It's where the model is forced to be efficient
- If too big: model memorizes; too small: model forgets
- Sweet spot = good generalization

Autoencoders: Architecture (cont.)

Decoder:

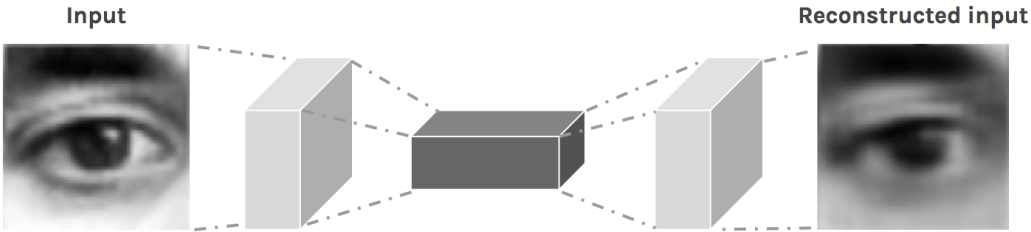
- Takes the bottleneck output
- Expands it back to the original input size
- Tries to make it as close as possible to the real input
- Learns to generate data from compressed features
- *Example*: Converts 32 numbers back to a 784-pixel image

Autoencoders: Architecture (cont.)

Component	Description
Encoder	Transforms the input data into a compressed latent representation using layers such as Dense, Convolutional, or Recurrent layers.
Latent Space (Bottleneck)	The central compact representation of the input. Forces the model to learn the most essential features.
Decoder	Reconstructs the input from the latent space. Typically mirrors the encoder's structure in reverse.
Loss Function	Measures the difference between input and output (e.g., Mean Squared Error, Binary Cross-Entropy). Drives learning during training.
Training	Involves minimizing reconstruction error using gradient-based optimization methods such as SGD or Adam.

Architecture Summary Table

Autoencoders: Architecture (cont.)



Sample Autoencoder

Source: J. Despois, "Autoencoders — Deep Learning bits #1", HackerNoon, 2017.

Autoencoders: Interactive Demo

https://douglasduhaime.com/posts/visualizing-latent-spaces.html

Autoencoders: **As Generative Models**

Autoencoders as Generative Models

- Autoencoders project data into a latent space Z.
- The latent space is not necessarily continuous or structured.

Autoencoders as Generative Models

- Autoencoders project data into a latent space Z.
- The latent space is not necessarily continuous or structured.
- *What if we sample a new embedding vector from Z and then have the decoder reconstruct the image from it?*

Autoencoders as Generative Models

- Autoencoders project data into a latent space Z.
- The latent space is not necessarily continuous or structured.
- *What if we sample a new embedding vector from Z and then have the decoder reconstruct the image from it?*
- **Does not work**. Autoencoders just learn a function that maps input to output. The learned latent space is too discontinuous to work as a generative model.

Autoencoders as Generative Models

- Autoencoders project data into a latent space Z.
- The latent space is not necessarily continuous or structured.
- *What if we sample a new embedding vector from Z and then have the decoder reconstruct the image from it?*
- **Does not work**. Autoencoders just learn a function that maps input to output. The learned latent space is too discontinuous to work as a generative model.
- Sampling randomly from the latent space may result in invalid outputs.
- They do not maximize the likelihood of the data.

Autoencoders as Generative Models

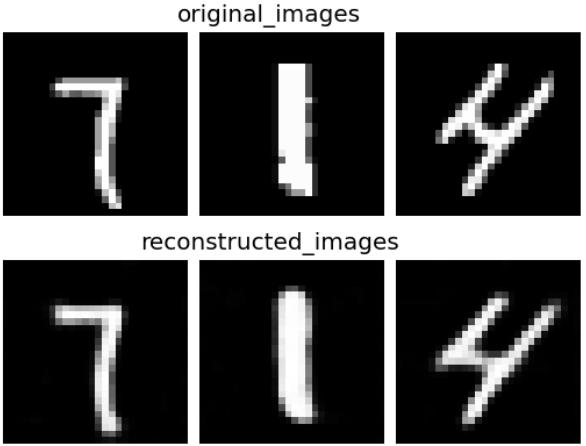


Image reconstruction with autoencoder trained on MNIST digits

Autoencoders as Generative Models (cont.)

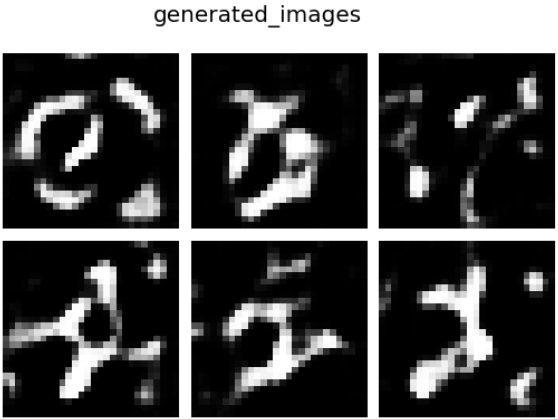


Image generation with autoencoder trained on MNIST digits. Encoding vector sampled from latent space Z and then passed to the decoder.

Autoencoders: **Applications**

Applications of Autoencoders

- Dimensionality Reduction
- Super-Resolution
- Colorization
- Anomaly Detection
- Denoising

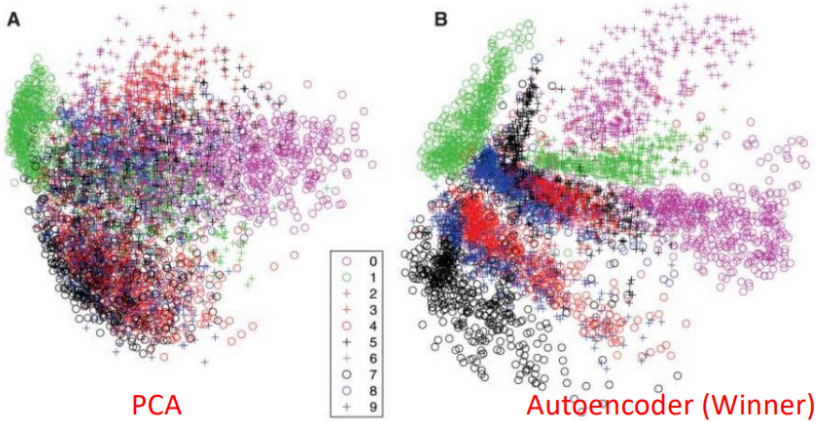
Dimensionality Reduction

Autoencoders help us reduce the number of features in our data, similar to PCA, but in a more flexible way.

- The encoder takes big, complex data and squeezes it into a smaller, simpler form.
- This smaller version keeps the important information.
- Great for making it easier to see patterns or to prepare data for other machine learning tasks.

Example: We can use autoencoders to turn images (like MNIST digits) into just 2 or 3 numbers, so we can plot and explore them easily.

Dimensionality Reduction (cont.)



2-D codes for MNIST digits produced by PCA (left) vs. the 2-unit code layer of a deep autoencoder (right). Source: Hinton & Salakhutdinov, *Science*, 2006.

Super-Resolution

Autoencoders can be used to reconstruct high-resolution images from their low-resolution counterparts.

- Input: Low-resolution image.
- Output: High-resolution image.
- Often implemented with convolutional layers for spatial pattern learning.

Use Case: Enhancing medical images, satellite images, or upscaling low-resolution photos.

Super-Resolution (cont.)



Image super-resolution using Autoencoders

Source: A. J. Champandard, Neural Enhance, 2016.

Image Colorization

Autoencoders can learn to predict color information for grayscale images.

- Input: Grayscale image (1 channel).
- Output: Colorized image (3 channels: RGB).
- Requires learning semantic and contextual relationships to apply realistic colors.

Use Case: Restoring historical black-and-white photos.

Image Colorization (cont.)

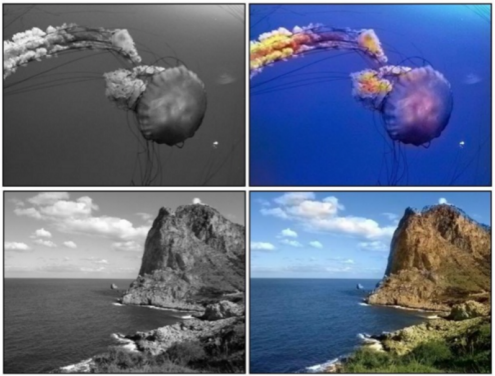


Image colorization using Autoencoders

Source: R. Zhang, P. Isola & A. A. Efros, "Colorful Image Colorization", ECCV 2016, teaser figure.

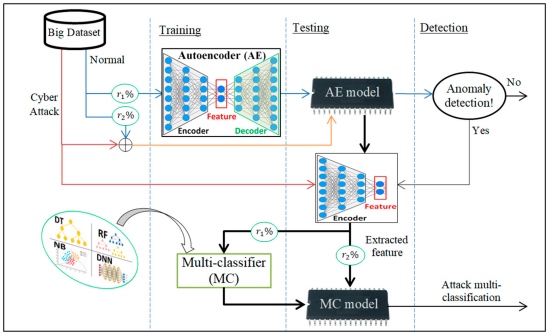
Anomaly Detection

- Autoencoders are trained to reconstruct input data.
- During training, the model learns to compress and decompress data that is similar to the training set (normal data).
- When presented with anomalous (outlier) data, the reconstruction error increases, as the autoencoder cannot effectively reconstruct unseen patterns.

Use Cases:

- Fraud detection in financial transactions
- Fault detection in industrial systems
- Intrusion detection in cybersecurity
- Medical anomaly detection (e.g., rare diseases in imaging)

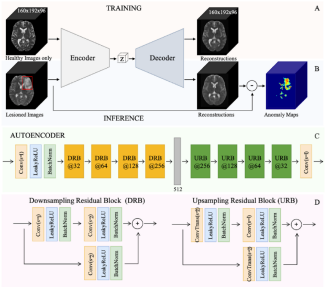
Anomaly Detection (cont.)



Structure of the deep autoencoder (AE) for anomaly detection and feature extraction for multi-classification of cyber-attacks.

Source: Alaghbari, Lim, Md Saad & Yong, "Deep Autoencoder-Based Integrated Model for Anomaly Detection and Efficient Feature Extraction in IoT Networks," *IoT*, 4(3), 2023, Fig. 2.

Anomaly Detection (cont.)



Autoencoder-based anomaly detection framework: A) Training the autoencoder for modeling the distribution of healthy brain anatomy. B) In the inference phase, anomaly maps reveal lesions by subtracting their reconstruction from the input image. C and D) Details of the proposed autoencoder.

Source: Luo, Xie, Gao, Zheng, Chen & Sun, "Unsupervised anomaly detection in brain MRI: Learning abstract distribution from massive healthy brains," *Computers in Biology and Medicine*, 154, 2023, Fig. 1.

Denoising

Key Idea:

- Train the autoencoder to reconstruct the original, clean input from a corrupted (noisy) version.
- The encoder learns robust features that capture the underlying structure of the data, ignoring noise.

Training Process:

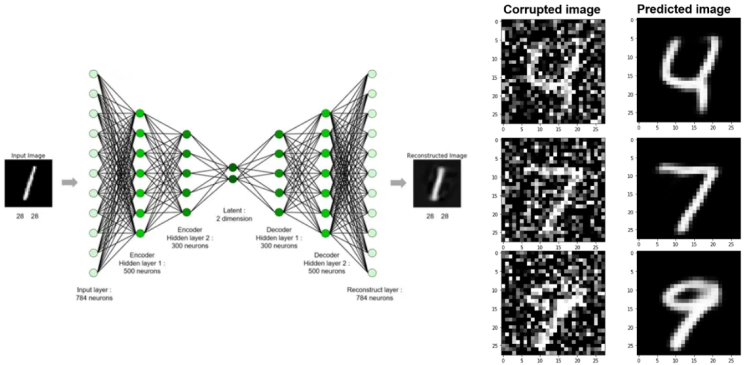
1. Add noise to the input data (e.g., Gaussian noise, salt-and-pepper noise).
2. Feed the noisy input to the autoencoder.
3. Compute the loss between the output and the original clean input.
4. Update the network weights to minimize this loss.

Denoising (cont.)

Use Cases:

- Image denoising: Removing noise from photographs or medical images.
- Speech enhancement: Improving audio quality by filtering out background noise.
- Preprocessing step: Providing cleaner data for downstream tasks such as classification or segmentation.

Denoising (cont.)



Example of denoising using an autoencoder.

Source: https://wikidocs.net/193827

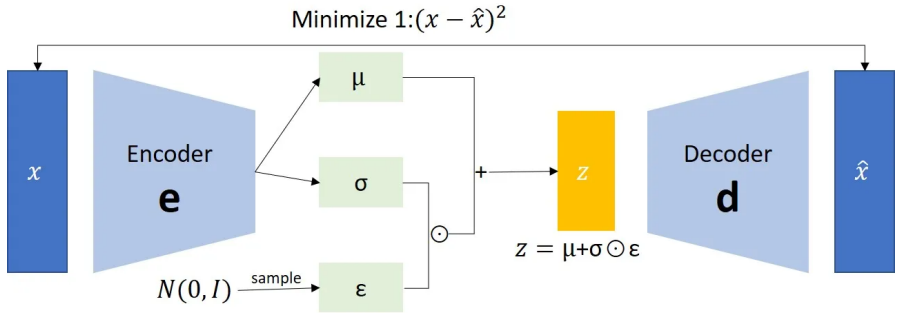
Autoencoders: **Challenges**

Challenges

- May just copy input without real learning
- Needs careful tuning of bottleneck size
- Sometimes blurry outputs (especially basic autoencoders)
- Cannot reliably generate new data; VAEs (covered next) and GANs (covered later in the course) address this

Variational Autoencoders (VAE)

Variational Autoencoders: **Motivation**



Minimize 2: $\frac{1}{2}\sum_{i=1}^{N}(\exp(\sigma_i) - (1+\sigma_i) + \mu_i{}^2)$

[Source: https://miro.medium.com/v2/resize:fit:4800/format:webp/1*r1R0cxCnErWgE0P4Q-hI0Q.jpeg]

Why do we need VAEs?

- Traditional autoencoders are:
  - deterministic
  - lack generative capabilities
  - do not model uncertainty in latent space
- Need a compact, meaningful representation (latent space)
- We want to learn a probabilistic model of the data
- We want to generate new data samples similar to training data

Variational Autoencoders: **Learning Outcomes**

⇒ Learning Outcomes

By the end of this session, you will be able to:

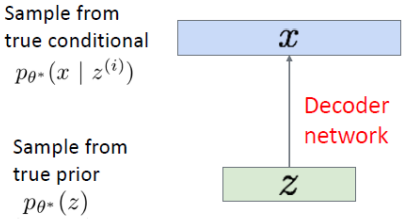
- Explain what a VAE is and why we use it
- Describe how VAEs create and use hidden (latent) spaces
- Understand how VAEs learn from data
- Understand the ELBO and its role
- See how we train VAEs using special tricks
- Recognize different types of VAEs and some common challenges

VAE: **Introduction**

From a Point to a Distribution

- An **autoencoder** learns a mapping: each input x lands on **one point** z in latent space. We just saw why sampling that space fails: nothing controls what lies *between* the points.
- A **VAE** changes what the encoder outputs: instead of a point, it predicts the parameters of a **distribution**, a mean $\mu(x)$ and a standard deviation $\sigma(x)$, and z is **sampled** from $\mathcal{N}(\mu(x), \sigma^2(x))$.
- Decoding from a sampled **region** rather than a fixed point forces nearby latents to decode to similar outputs: the space becomes smooth by construction.
- Everything that follows (reparameterization, the KL term, the ELBO) exists to make this simple idea trainable and useful.

- We want a **generative model**: given a latent **z** drawn from a prior, it outputs a new sample from the data distribution.
- To make the latent space Z continuous, let's choose it to be Gaussian.
- So now, we want to estimate the true parameters θ^* of this generative model.



Source: Stanford CS231n, Lecture 13 "Generative Models" (Fei-Fei Li, Justin Johnson, Serena Yeung), 2017.

Introduction (cont.)

- **Probabilistic Encoder:** Instead of encoding an input x to a fixed point z, the encoder learns a distribution over the latent variable z conditioned on the input:

$$q(z \mid x)$$

Typically, this is a multivariate Gaussian with parameters $\mu(x)$ and $\sigma(x)$ learned by a neural network.

- **Probabilistic Decoder:** Given a sample z from the latent distribution, the decoder reconstructs the input by modeling:

$$p(x \mid z)$$

This allows for generating new data by sampling from the latent space.

- **Latent Space:**
  - The model learns a smooth, continuous space for z.
  - Similar data points end up close together in this space.
  - This helps the model understand the main patterns in the data.

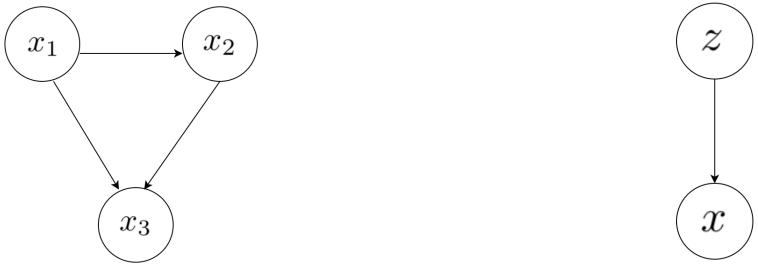
Why Use VAEs?

- Enable generative modeling: generate new samples by sampling from the latent distribution.
- Learn smooth, structured, and interpretable latent representations.
- Provide a principled probabilistic framework for inference and generation.
- Trained by maximizing the **Evidence Lower Bound (ELBO)**, which we derive in the following sections.

VAE: **Latent Variable Models**

Latent Variable Models

- **Fully observed models**: every random variable is observed.
- **Latent Variable Models (LVMs)**: Some random variables are hidden – we do not get to observe them.



Source: UC Berkeley CS294-158 Deep Unsupervised Learning, Spring 2020, L4 Latent Variable Models (Abbeel, Chen, Ho, Srinivas, Li, Yan), slide 2.

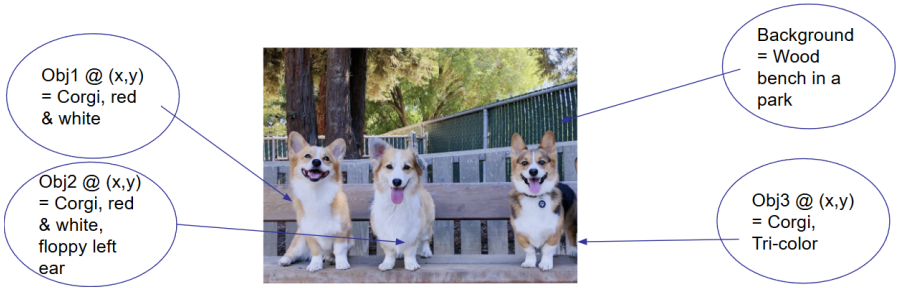
Latent Variable Models (cont.)

Latent Variable Models:

- **Definition**: Models that assume the data is generated from some unobserved (latent) variables.
- **Goal**: Learn a mapping from observed data x to latent variables z and vice versa.
- **Generative Process**:
  - Sample latent variable z from a prior distribution $p(z)$.
  - Generate data x from the latent variable using a likelihood function $p(x|z)$.
- **Inference Problem**: Given observed data x, infer the posterior distribution $p(z|x)$.
- **Challenge**: Direct computation of posterior $p(z|x)$ is often intractable, leading to the need for approximations.

Why Latent Variable Models?

- Simpler, lower-dimensional representations of data often possible
  - Latent variable models hold the promise of automatically identifying those hidden representations



Source: UC Berkeley CS294-158 Deep Unsupervised Learning, Spring 2020, L4 Latent Variable Models (Abbeel, Chen, Ho, Srinivas, Li, Yan).

Why Latent Variable Models? (cont.)

- Autoregressive (AR) models are slow to sample because all pixels (observation dimensions) are assumed to be dependent on each other.
- We can make part of the observation space independent *conditioned on some latent variables*.
- Latent variable models *can* have faster sampling by exploiting statistical patterns.

Why Latent Variable Models? (cont.)

- Sometimes, it's possible to design a latent variable model with an understanding of the causal process that generates data
- In general, we don't know what the latent variables are and how they interact with observations
- Most popular models make few assumptions about what the latent variables are
- Best way to specify latent variables is still an active area of research

Latent Variable Model

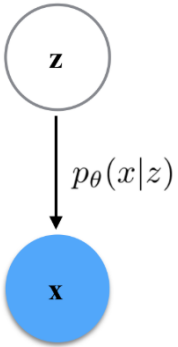
- **Sample**

$$z \sim p_Z(z)$$
$$x \sim p_\theta(x \mid z)$$

- **Evaluate likelihood** $p_\theta(x) = \sum_z p_Z(z) p_\theta(x \mid z)$

- **Train** $\max_\theta \sum_i \log p_\theta(x^{(i)}) = \sum_i \log \sum_z p_Z(z) p_\theta(x^{(i)} \mid z)$

- **Representation** $x \rightarrow z$



Source: UC Berkeley CS294-158 Deep Unsupervised Learning, Spring 2020, L4 Latent Variable Models (Abbeel, Chen, Ho, Srinivas, Li, Yan).

VAE: **Variational Inference**

Variational Inference

- Now, how to train this model?

Variational Inference

- Now, how to train this model?
- Learn model parameters by maximizing the likelihood of the training data (maximum likelihood, as for the models seen so far).

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Variational Inference

- Now, how to train this model?
- Learn model parameters by maximizing the likelihood of the training data (maximum likelihood, as for the models seen so far).

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- But there is a problem here. It is intractable to compute $p(x|z)$ for every z!

Variational Inference

- Now, how to train this model?
- Learn model parameters by maximizing the likelihood of the training data (maximum likelihood, as for the models seen so far).

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- But there is a problem here. It is intractable to compute $p(x|z)$ for every z!
- Intuitively, need to figure out which z corresponds to each x in the dataset, but such mapping is unknown.

Variational Inference

- Now, how to train this model?
- Learn model parameters by maximizing the likelihood of the training data (maximum likelihood, as for the models seen so far).

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- But there is a problem here. It is intractable to compute $p(x|z)$ for every z!
- Intuitively, need to figure out which z corresponds to each x in the dataset, but such mapping is unknown.
- This also makes posterior density $p(z|x)$ intractable because it depends on $p_\theta(x)$

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

Solution

- Let's approximate $p(z|x)$ with another distribution $q(z)$
- If $q(z)$ is a tractable distribution e.g. Gaussian distribution
- **Approach**: We can adjust parameters of $q(z)$ and make it as close to $p(z|x)$, i.e. $q(z) \approx p(z|x)$.
- **Goal**: Minimize the Kullback-Leibler (KL) divergence $KL(q||p)$.

Variational Inference (cont.)

$$
\begin{aligned}
KL(q(z)||p(z|x)) &= -\sum q(z) \log \frac{p(z|x)}{q(z)} \\
&= -\sum q(z) \log \frac{\frac{p(x,z)}{p(x)}}{q(z)} \\
&= -\sum q(z) \log \left(\frac{p(x,z)}{q(z)} \frac{1}{p(x)} \right) \\
&= -\sum q(z) \left[\log \frac{p(x,z)}{q(z)} - \log\ p(x) \right] \\
&= -\sum_z q(z) \log \frac{p(x,z)}{q(z)} + \log\ p(x) \sum_z q(z) \\
&= -\sum_z q(z) \log \frac{p(x,z)}{q(z)} + \log\ p(x) \quad \because \sum_z q(z) = 1
\end{aligned}
$$

Variational Inference (cont.)

- $$KL(q(z)||p(z|x)) = -\sum_z q(z) \log \frac{p(x,z)}{q(z)} + \log\ p(x)$$

- We can also write above equation as:

$$\log\ p(x) = KL(q(z)||p(z|x)) + \sum_z q(z) \log \frac{p(x,z)}{q(z)}$$

Variational Inference (cont.)

- Given x , $\log\ p(x)$ is a constant
- $KL(q(z)||p(z|x))$ is the quantity we wanted to minimize
- Assume $L = \sum_z q(z) \log \frac{p(x,z)}{q(z)}$, then

$$\text{constant} = KL + L$$

$$L \leq \log\ p(x) \quad \because KL \geq 0$$

- **Instead of minimizing KL we can maximise L**

VAE: **Evidence Lower Bound (ELBO)**

Evidence Lower Bound (ELBO)

What is ELBO?:

- Allows us to optimize the model.
- It provides a lower bound on the log-likelihood of the data, which we aim to maximize during training.
- The ELBO consists of two main components:
  - **Reconstruction term**: Measures how well the model can reconstruct the input data from the latent representation.
  - **Regularization term**: Encourages the learned latent distribution to be close to a prior distribution (usually Gaussian).

Evidence Lower Bound (ELBO) (cont.)

- Mathematically, the ELBO can be expressed as:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) \| p(z))$$

  where:

  - $q_\phi(z|x)$ is the approximate posterior distribution (encoder).
  - $p_\theta(x|z)$ is the likelihood of the data given the latent variable (decoder).
  - D_{KL} is the Kullback-Leibler divergence between the approximate posterior and the prior distribution $p(z)$.

- The goal is to maximize the ELBO with respect to the model parameters θ and ϕ.
- By maximizing the ELBO, we ensure that the model learns a meaningful latent representation while also being able to generate new samples.

Evidence Lower Bound (ELBO) (cont.)

Looking at Lower bound L

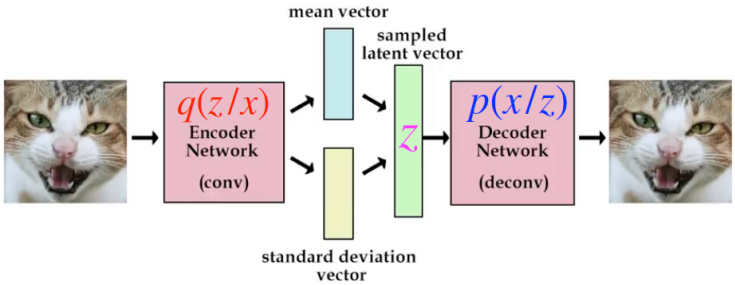
$$
\begin{aligned}
L &= \sum_z q(z) \log \frac{p(x,z)}{q(z)} \\
&= \sum_z q(z) \log \frac{p(x|z)p(z)}{q(z)} \\
&= \sum_z q(z) \left[\log\ p(x|z) + \log \frac{p(z)}{q(z)} \right] \\
&= \underbrace{\sum_z q(z) \log\ p(x|z)}_{\text{Expectation } E_{q(z)}(\log\ p(x|z))} + \underbrace{\sum_z q(z) \log \frac{p(z)}{q(z)}}_{-KL(q(z)||p(z))}
\end{aligned}
$$

So,

$$
L = E_{q(z)}(\log\ p(x|z)) - KL(q(z)||p(z))
$$

Evidence Lower Bound (ELBO) (cont.)

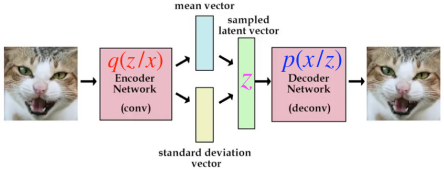
- $E_{q(z)}(\log\, p(x|z))$ is conceptually reconstruction
- We choose the prior $p(z)$ to be the standard normal distribution



Variational Autoencoder Architecture

Source: K. Frans, "Variational Autoencoders Explained", 2016; the $q(z|x)$, $p(x|z)$ and z labels were added later.

Evidence Lower Bound (ELBO) (cont.)



Source: K. Frans, "Variational Autoencoders Explained", 2016.

$$p(x|z) \propto e^{-|x-\hat{x}|^2}$$

$$\log e^{-|x-\hat{x}|^2} = -|x-\hat{x}|^2$$

$$L = E_{q(z)}(-|x-\hat{x}|^2) - KL(q(z)||p(z))$$

$$\min |x-\hat{x}|^2 + KL(q(z|x)||\mathcal{N}(0,1))$$

$$\min |x-\hat{x}|^2 - 0.5 * (1 + \log \sigma^2 - \sigma^2 - \mu^2)$$

For full derivation of KL Loss, read here

VAE: **Reparameterization Trick (Pathwise Derivative Estimator)**

The Problem: Sampling Blocks Gradients

- The forward pass is clear: encoder outputs $\mu(x)$ and $\sigma(x)$, we **sample** $z \sim \mathcal{N}(\mu(x), \sigma^2(x))$, and the decoder reconstructs from z.
- Training needs $\partial\mathcal{L}/\partial\mu$ and $\partial\mathcal{L}/\partial\sigma$, but the gradient path runs **through the sampling node**, and "draw a random number" is not a differentiable operation.
- So the vanilla design of just predicting μ and σ cannot be trained end-to-end by backpropagation as it stands.
- Score-function (REINFORCE) estimators could work around this, but their gradients are extremely noisy.
- The fix is to **move the randomness out of the way**: the reparameterization trick.

Reparameterization Trick

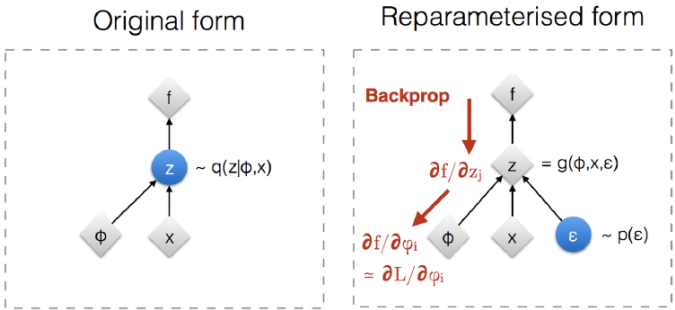
Problem: Cannot backpropagate through stochastic sampling.

Solution: Reparameterize z as:

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

Benefit: Enables gradient-based optimization by making the sampling operation differentiable.

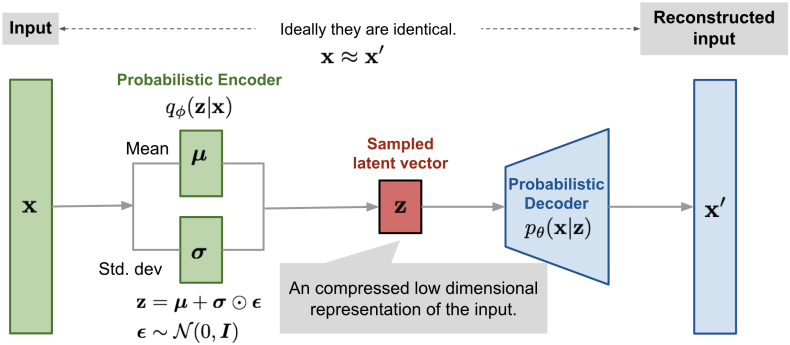
Reparameterization Trick (cont.)



[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Reparameterization trick to make backpropagation possible

Reparameterization Trick (cont.)



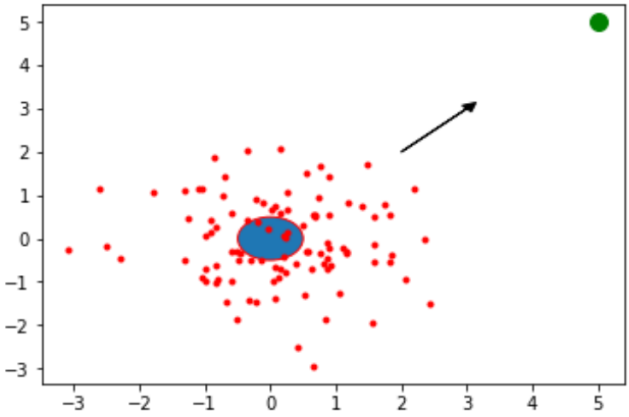
Variational Autoencoder with reparameterization trick

Source: L. Weng, "From Autoencoder to Beta-VAE", 2018.

Pathwise Derivative - Toy Problem

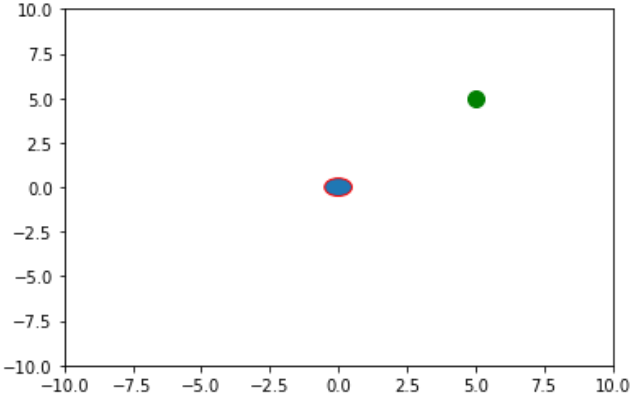
Learn $\mu \in \mathbb{R}^2$ to minimize the objective below to reach the green point

$$\mathcal{L} = \mathbb{E}_{x \sim N(\mu, I)} \left\| x - \begin{bmatrix} 5 \\ 5 \end{bmatrix} \right\|_2^2$$



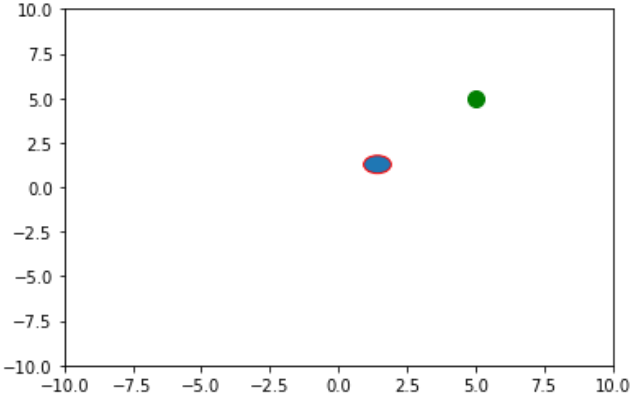
Source: UC Berkeley CS294-158 Deep Unsupervised Learning, Spring 2020, L4 Latent Variable Models (Abbeel, Chen, Ho, Srinivas, Li, Yan).

Pathwise Derivative - Toy Problem (cont.)



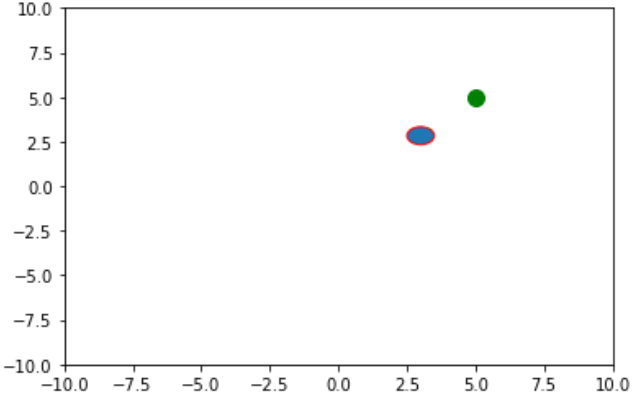
Source: UC Berkeley CS294-158 Deep Unsupervised Learning, Spring 2020, L4 Latent Variable Models (Abbeel, Chen, Ho, Srinivas, Li, Yan).

Pathwise Derivative - Toy Problem (cont.)



Source: UC Berkeley CS294-158 Deep Unsupervised Learning, Spring 2020, L4 Latent Variable Models (Abbeel, Chen, Ho, Srinivas, Li, Yan).

Pathwise Derivative - Toy Problem (cont.)



Source: UC Berkeley CS294-158 Deep Unsupervised Learning, Spring 2020, L4 Latent Variable Models (Abbeel, Chen, Ho, Srinivas, Li, Yan).

VAE: **Loss Function**

Loss Function Breakdown

Total Loss:

$$L = \text{Reconstruction Loss} + \text{KL Divergence}$$

Reconstruction Loss:

- Measures how well $\hat{x}$ matches x.
- Common choices: Mean Squared Error (MSE) or Binary Cross-Entropy.

KL Divergence:

- Measures how much $q(z \mid x)$ diverges from the prior $p(z)$.
- Encourages the latent space to follow a standard normal distribution.

Loss Function Breakdown (cont.)

What if we trained with reconstruction only?

- The easiest way to reconstruct well is to be **deterministic**: the encoder shrinks $\sigma \to 0$ and the model degenerates into a plain autoencoder.
- Nothing constrains **where** the codes live: clusters drift far apart with empty space between them, and the decoder is undefined on the gaps.
- Sampling from $\mathcal{N}(0, I)$ then lands in regions the decoder has never seen; generation fails exactly as it did for the autoencoder.
- The KL term is what prevents both failures: it keeps σ from collapsing and pulls all codes toward a shared, densely covered region around the origin.

Loss Function Breakdown (cont.)



Reconstruction only: codes scatter and leave the prior sitting in a hole (left). KL only: every posterior collapses onto the prior and the classes stop being separable (middle). Both terms together: well-separated clusters packed around the prior, so every sample decodes sensibly (right).

Source: I. Shafkat, "Intuitively Understanding Variational Autoencoders", Towards Data Science, 2018; composite via J. Jordan, "Variational autoencoders", 2018.

VAE: **Understanding KL Divergence**

What Is KL Divergence?

Definition: the Kullback-Leibler divergence measures how different a distribution p is from a reference distribution q:

$$D_{\mathrm{KL}}(p \,\|\, q) \;=\; \sum_{x} p(x) \log \frac{p(x)}{q(x)} \;=\; \mathbb{E}_{x\sim p}[\log p(x) - \log q(x)]$$

Properties:

- $D_{\mathrm{KL}}(p \,\|\, q) \geq 0$, with equality exactly when $p = q$.
- **Asymmetric:** $D_{\mathrm{KL}}(p \,\|\, q) \neq D_{\mathrm{KL}}(q \,\|\, p)$ in general, so it is not a distance metric.
- **Intuition:** with $\log_2$, it is the expected number of *extra bits* needed to encode samples from p using a code designed for q.

What Is KL Divergence? (cont.)

KL divergence vs. cross-entropy: the two are directly related:

$$\underbrace{H(p, q)}_{\text{cross-entropy}} = \underbrace{H(p)}_{\text{entropy of } p} + \underbrace{D_{\mathrm{KL}}(p \,\|\, q)}_{\text{KL divergence}}$$

- In **classification**, p is the fixed label distribution (one-hot), so $H(p)$ is a constant: minimizing cross-entropy and minimizing KL are *the same thing*. That is why classifiers use cross-entropy.
- In a **VAE**, both $q(z \mid x)$ and the comparison to the prior depend on the model, and we need the divergence itself as a regularizer, not a classification loss. So the KL term appears explicitly.

What Is KL Divergence? (cont.)

KL numerically (Gaussians): for a 1-D Gaussian against the standard normal prior there is a closed form:

$$D_{\mathrm{KL}}\left(\mathcal{N}(\mu, \sigma^2) \,\|\, \mathcal{N}(0,1)\right) = \tfrac{1}{2}\left(\mu^2 + \sigma^2 - 1 - \log \sigma^2\right)$$

μ	σ	D_{KL}	
0	1	0	(matches the prior exactly)
2	1	2.0	(penalizes drifting away)
0	0.1	≈ 1.81	(penalizes collapsing to a point)
0	2	≈ 0.81	(penalizes spreading too wide)

- Summed over dimensions this is exactly the VAE regularizer from the ELBO section: $-\frac{1}{2}\sum_i \left(1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\right)$.
- Note the two-sided pressure: drifting from the prior *and* becoming deterministic ($\sigma \to 0$) are both punished.

VAE: **Variants and Extensions**

Variants and Extensions

β-VAE:

- Introduces a hyperparameter β to control the trade-off between reconstruction and regularization.
- Encourages disentangled representations.

Conditional VAE (CVAE):

- Incorporates additional information y (e.g., class labels) into the encoder and decoder.
- Enables generation of data conditioned on y.

Discrete VAE:

- Utilizes discrete latent variables.
- Techniques like Gumbel-Softmax are used for differentiable sampling.

Disentangled Variational Autoencoders (β-VAEs)

- **Basic Idea**: Different neurons in latent space should be uncorrelated, i.e. they all try to learn something different about input data.
- **Implementation**:

$$\mathcal{L}(\theta, \phi; x, z, \beta) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta\, D_{\mathrm{KL}}(q_\phi(z|x) \,\|\, p(z))$$

- Increasing β forces the variational autoencoder to encode the information in only a few latent variables

Disentangled Variational Autoencoders (β-VAEs) (cont.)



Azimuthal rotation in β-VAEs and simple VAEs. β-VAEs produce more disentangled rotation, whereas some other features also change in simple VAEs.

Source: Higgins et al., "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," ICLR 2017, Fig. 1a.

VAE: **Results**

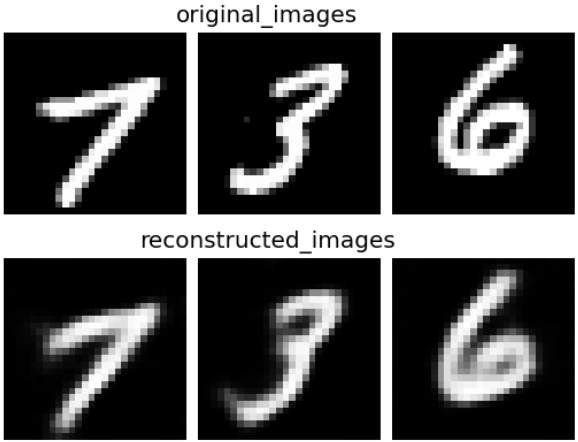


Image reconstruction with variational autoencoders on MNIST digits dataset

Results (cont.)

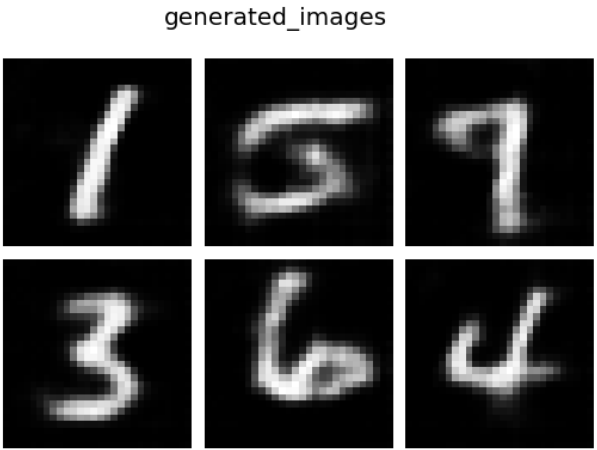


Image generation with variational autoencoders on MNIST digits dataset. Sample an encoding vector from $\mathcal{N}(0, I)$ and pass it through the decoder

Results (cont.)



Image generation with variational autoencoders on the CIFAR-10 32×32 dataset

Results (cont.)



Faces generated with a deep feature consistent VAE (https://github.com/houxianxu/DFC-VAE)

VAE: **Applications**

Where Are VAEs Used?

- **Generation and interpolation:** sample new data from the prior; walk the smooth latent space to morph between examples or do attribute arithmetic.
- **Anomaly detection:** train on normal data; anomalous inputs reconstruct poorly, so the reconstruction error becomes an anomaly score (industrial inspection, medical imaging, fraud).
- **Representation learning:** the latent code is a compact, structured feature vector for downstream tasks; β-VAEs push this toward disentangled factors.
- **Scientific design:** molecule and drug design search the continuous latent space instead of discrete chemical space.
- **Compression inside larger systems:** the autoencoder of **latent diffusion** (Stable Diffusion) is a VAE; it provides the compressed space the diffusion model works in (covered later in the course).

VAE: **Perceptual Loss**

Pixel Losses See Pixels, Not Content

- MSE compares images **pixel by pixel**: shift an image by one pixel and the error explodes, yet a blurry average scores well.
- Under uncertainty, the reconstruction that minimizes MSE is the **mean** of all plausible images, and the mean of sharp images is a blurry image.
- **Perceptual loss** (Johnson et al., 2016): compare images in the **feature space of a pretrained network** (e.g. VGG) instead of pixel space:

$$\mathcal{L}_{\text{perc}}(x, \hat{x}) = \sum_{l} \lambda_l \left\| \phi_l(x) - \phi_l(\hat{x}) \right\|_2^2$$

  where ϕ_l are feature maps at chosen layers l.
- Deep features respond to textures, edges, and object parts, so the loss judges **content similarity** rather than exact pixel agreement. **LPIPS** (Zhang et al., 2018) calibrates this distance against human judgments.

Perceptual Losses in VAEs

- **Deep Feature Consistent VAE** (Hou et al., 2017): replace the pixel reconstruction term with a perceptual one; the KL term stays unchanged. Reconstructions keep fine detail (the face results shown earlier use exactly this).
- The loss becomes:

$$\mathcal{L} \;=\; \mathcal{L}_{\text{perc}}(x, \hat{x}) \;+\; D_{\text{KL}}(q(z \mid x) \,\|\, p(z))$$

- **Modern practice:** the autoencoders behind large generative systems (VQGAN, Stable Diffusion's VAE) combine pixel, **perceptual (LPIPS)**, and **adversarial** losses (a learned discriminator penalizes unrealistic reconstructions; covered later in the course) to stay sharp at strong compression.
- **Trade-offs:** the objective is no longer a clean likelihood bound, and the pretrained network's biases decide what counts as "similar"; but for visual quality the gain is large.

VAE: **Two Problems on the Road to VQ-VAE**

Problem 1: The Averaging Problem

- The decoder must produce **one** output for each z, and the Gaussian likelihood (MSE) makes the optimal output the **average** of everything that z could correspond to.
- The KL term makes this worse: it deliberately **overlaps** the posteriors of different inputs, so one latent region is shared by several training images and decodes to their compromise.
- Result: the characteristic **blurriness** of VAE samples; sharp textures and fine details are averaged away.
- Perceptual losses treat the symptom; the multimodality itself calls for a more expressive latent structure.

Problem 2: The Prior Hole Problem

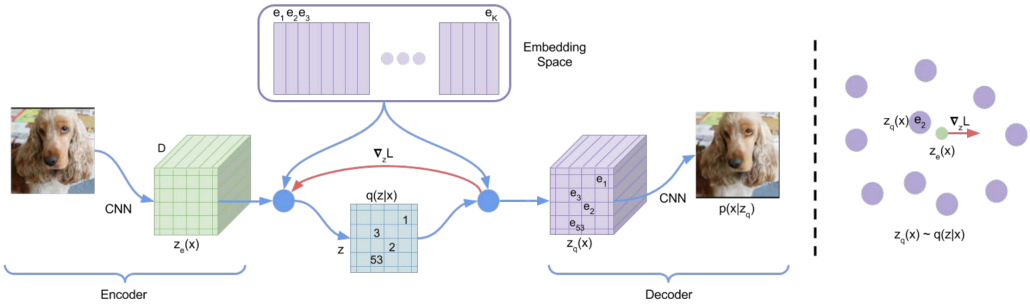
- Training only ever feeds the decoder latents drawn from the posteriors $q(z \mid x)$ of **real data**; generation samples from the **prior** $p(z) = \mathcal{N}(0, I)$.
- The KL term pushes each posterior toward the prior, but the **aggregate** of all posteriors never matches it exactly: some prior regions receive *no* training data. These are **prior holes**.
- A sample landing in a hole is decoded by a network that has never seen that region: outputs degrade or break entirely (the failure our latent-space figure showed for plain autoencoders, in a milder form).
- Related failure: **posterior collapse**; with a powerful decoder, the model can ignore z altogether and the latent space carries no information.
- Ways out: richer learned priors, hierarchical VAEs, or a different idea entirely: **discrete** latent codes with a **learned prior**. That is VQ-VAE.

VAE: **Vector Quantized VAE (VQ-VAE)**

VQ-VAE: Discrete Latent Codes

- **VQ-VAE** (van den Oord et al., 2017) replaces the Gaussian latent with a **codebook**: K learnable embedding vectors $e_1, \ldots, e_K$ (e.g. $K = 512$).
- The encoder produces a **grid** of vectors, and each one is **snapped to its nearest codebook entry**; an image becomes a small grid of discrete token indices (e.g. 32×32 for ImageNet).
- Why discrete?
  - Much of the world is naturally discrete: words, phonemes, object identities.
  - A code must commit to **one** codebook entry; there is no Gaussian averaging between concepts.
  - With a uniform prior over codes, the KL term is a **constant**: no reconstruction-vs-KL balancing act and no posterior collapse.

VQ-VAE: Architecture



Source: van den Oord, Vinyals & Kavukcuoglu, "Neural Discrete Representation Learning," NeurIPS 2017, Fig. 1.

VQ-VAE: Discrete Posterior

- A latent embedding space $e \in \mathbb{R}^{K \times D}$ where K is the size of the discrete latent space (K-way categorical distribution)
- Encoder output $z_e(x)$ is mapped to discrete latent code z by a nearest neighbour look-up using the shared embedding space e
- The posterior categorical distribution $q(z|x)$ probabilities are defined as one-hot as:

$$q(z = k|x) = \begin{cases} 1 & \text{for } k = \arg\min_j \|z_e(x) - e_j\|_2 \\ 0 & \text{otherwise} \end{cases}$$

VQ-VAE: Quantization and Loss

- This model can be seen as a VAE in which the proposal distribution $q(z = k|x)$ is deterministic.
- The representation $z_e(x)$ is passed through the discretization bottleneck followed by mapping onto the nearest element of embedding e

$$z_q(x) = e_k, \qquad \text{where} \qquad k = \arg\min_j \|z_e(x) - e_j\|_2$$

$$\mathcal{L} = -\log p(x \mid z_q(x)) + \|\text{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \text{sg}[e]\|_2^2$$

sg stands for the **stopgradient** operator that is defined as identity at forward computation time and has zero partial derivatives, thus effectively constraining its operand to be a non-updated constant.

- The embedding space is learned via Vector Quantization (VQ), whose objective uses the l_2 error to move the embedding vectors e_i towards the encoder outputs $z_e(x)$

The Gradient Problem: arg min Is Not Differentiable

- Picking the nearest codebook entry is a **hard selection**: the mapping $z_e \mapsto z_q$ is piecewise constant, so its gradient is **zero almost everywhere** (and undefined on the boundaries).
- Backpropagation from the decoder therefore stops dead at the quantization step; the encoder would never learn.
- **Straight-through estimator:** in the forward pass use z_q; in the backward pass **copy the gradient** $\nabla_{z_q}\mathcal{L}$ straight to z_e, as if quantization were the identity.
- This is an approximation, but a reasonable one because the commitment loss keeps z_e close to the codebook entry it maps to.
- Same family of tricks as the discrete VAE's Gumbel-Softmax from the variants slide; VQ-VAE's version is simpler and works at scale.

The Three Terms of the VQ-VAE Loss

$$\mathcal{L} = \underbrace{-\log p(x \mid z_q(x))}_{\text{1. reconstruction}} + \underbrace{\|\mathrm{sg}[z_e(x)] - e\|_2^2}_{\text{2. codebook loss}} + \underbrace{\beta \|z_e(x) - \mathrm{sg}[e]\|_2^2}_{\text{3. commitment loss}}$$

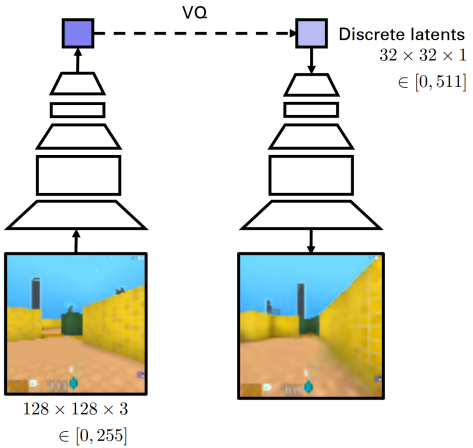
- **1. Reconstruction:** trains the **decoder** directly and the **encoder** through the straight-through gradient copy.
- **2. Codebook loss:** the stop-gradient freezes the encoder output, so only the **codebook entries** move toward their assigned encoder outputs (EMA updates are an alternative).
- **3. Commitment loss:** the stop-gradient freezes the codebook, so only the **encoder** moves; it commits to its entry instead of drifting; $\beta \approx 0.25$.
- Each term has one job; the two stop-gradients keep the jobs separate.

Two-Stage Training: Codebook First, Prior Second

- **Stage 1** trains encoder, decoder, and codebook with the loss above. The prior over codes is assumed **uniform**, so the KL term is constant and simply dropped.
- But sampling a uniformly random grid of tokens decodes to noise: real images use **structured** combinations of codes.
- **Stage 2:** freeze everything from stage 1 and fit an **autoregressive prior** over the discrete code grid: PixelCNN for images, WaveNet for audio. To generate: sample a code grid from the prior, then decode it.
- **Why two stages?** It separates **perceptual compression** (stage 1) from **distribution modeling** (stage 2), and the prior works on a small token grid (32×32) instead of 128^2 pixels.
- This blueprint outlived VQ-VAE: VQ-VAE-2, VQGAN with transformers, DALL·E 1, and the tokenizer-plus-generator pattern of modern latent generative models all follow it.

VQ-VAE

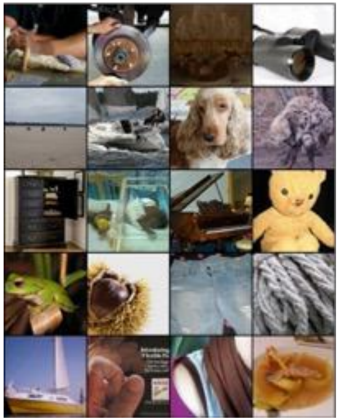
- For speech, images, and videos one can extract 1D, 2D, and 3D latent feature spaces respectively
- 32×32 latents for ImageNet, or $8 \times 8 \times 10$ for CIFAR10
- 1D sequences of latents drawn from a 512-entry codebook ($K = 512$) for VCTK and LibriSpeech



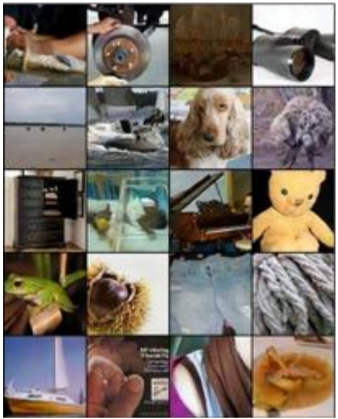
Source: A. van den Oord, "Neural Discrete Representation Learning," SANE 2017 talk slides (DeepMind).

VQ-VAE ImageNet Reconstruction

Original 128 x 128 images

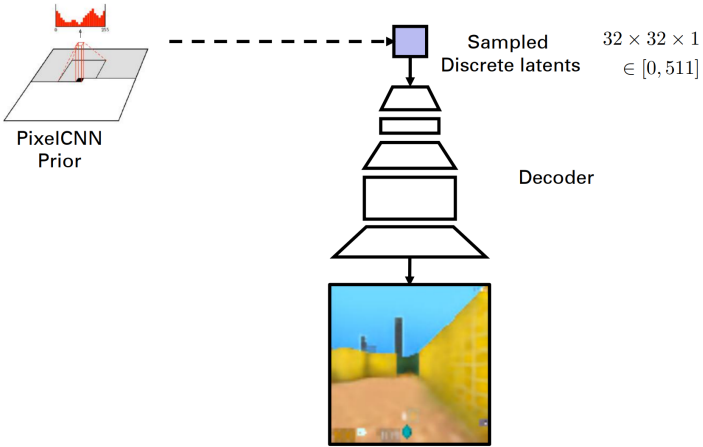


Reconstructions



Source: A. van den Oord, "Neural Discrete Representation Learning," SANE 2017 talk slides (DeepMind).

VQ-VAE Sampling



Source: A. van den Oord, "Neural Discrete Representation Learning," SANE 2017 talk slides (DeepMind).

VQ-VAE ImageNet Samples

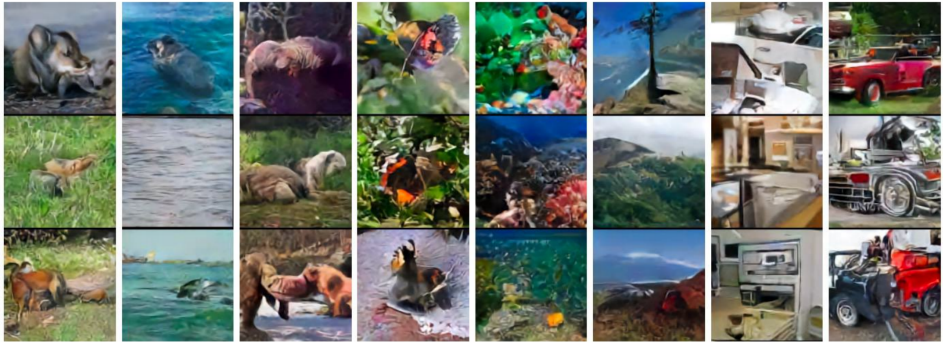
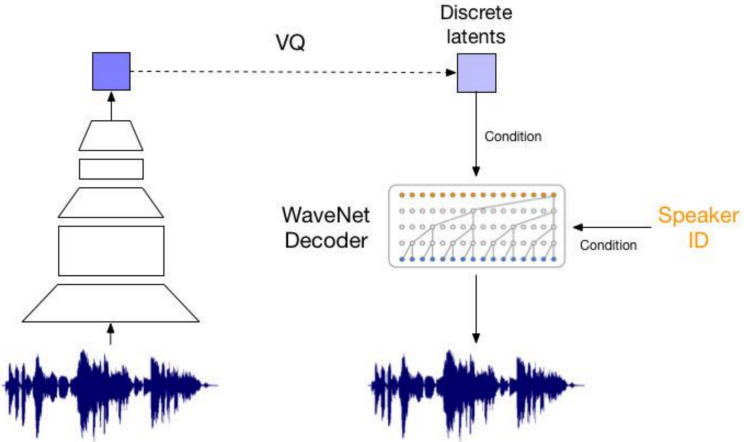


Figure 3: Samples (128x128) from a VQ-VAE with a PixelCNN prior trained on ImageNet images. From left to right: kit fox, gray whale, brown bear, admiral (butterfly), coral reef, alp, microwave, pickup.

Source: van den Oord, Vinyals & Kavukcuoglu, "Neural Discrete Representation Learning," NeurIPS 2017, Fig. 3.

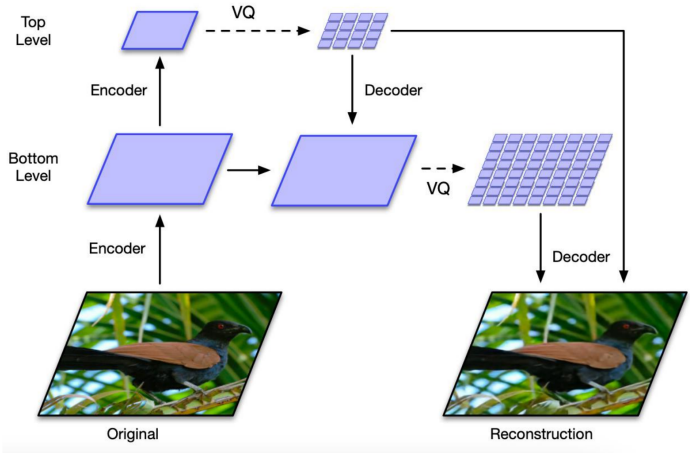
VQ-VAE Voice Style Transfer



Source: A. van den Oord, "Neural Discrete Representation Learning," SANE 2017 talk slides (DeepMind).

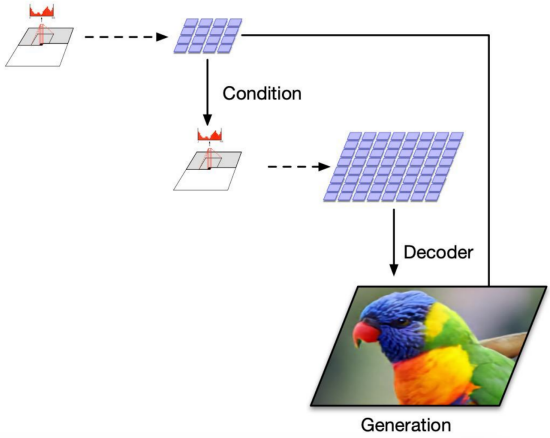
VQ-VAE 2

- The input 256×256 image is compressed to quantized latent maps of size 64×64 and 32×32
- The decoder reconstructs the image from the two latent maps.



Source: Razavi, van den Oord & Vinyals, "Generating Diverse High-Fidelity Images with VQ-VAE-2," NeurIPS 2019, Fig. 2a.

VQ-VAE-2 Sampling



Source: Razavi, van den Oord & Vinyals, "Generating Diverse High-Fidelity Images with VQ-VAE-2," NeurIPS 2019, Fig. 2b.

VQ-VAE-2 ImageNet Samples



Samples from the three-level hierarchical model trained on FFHQ 1024×1024. Source: Razavi, van den Oord & Vinyals, "Generating Diverse High-Fidelity Images with VQ-VAE-2," NeurIPS 2019, Fig. 6.

VAE: **Limitations and Challenges**

Limitations and Challenges

Limitations:

- **Gaussian Assumption:** The assumption that the latent variables follow a Gaussian distribution may not hold for all datasets.
- **Over-smoothing:** The model may produce overly smooth reconstructions, losing fine details in the data.
- **Sensitivity to Hyperparameters:** The performance of VAEs can be sensitive to the choice of hyperparameters, such as the weight of the KL divergence term.
- **Computational Complexity:** Training VAEs can be computationally expensive, especially for large datasets or complex models.
- **Evaluation Metrics:** Evaluating the quality of generated samples can be subjective and challenging, as traditional metrics may not capture the nuances of the data.

Limitations and Challenges (cont.)

Challenges:

- **Training Instability:** VAEs can be difficult to train, especially with complex datasets.
- **Balancing Reconstruction and Regularization:** Finding the right balance between reconstruction loss and KL divergence can be tricky.
- **Posterior Collapse:** In some cases, the model may ignore the latent variables, leading to poor representations.
- **Limited Expressiveness:** The Gaussian assumption for the latent space may not capture complex data distributions.
- **Disentanglement:** Achieving disentangled representations can be challenging, especially in high-dimensional spaces.

Limitations and Challenges (cont.)

Future Directions:

- **Improved Training Techniques:** Developing better optimization methods to stabilize training.
- **Advanced Architectures:** Exploring more complex latent variable models, such as Normalizing Flows or Hierarchical VAEs.
- **Better Regularization Techniques:** Investigating alternative regularization methods to improve disentanglement and representation quality.
- **Hybrid Models:** Combining VAEs with adversarial training to sharpen samples while keeping a structured latent space.
- **Application-Specific Variants:** Tailoring VAEs for specific applications, such as text or video generation.

Autoencoders: **References**

References: Autoencoders

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504-507.

[3] Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning* (pp. 1096-1103).

[4] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8), 1798-1828.

[5] Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning* (pp. 37-49).

References: Autoencoders (cont.)

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[7] Alaghbari, K. A., Lim, H.-S., Md Saad, M. H., & Yong, Y. S. (2023). Deep Autoencoder-Based Integrated Model for Anomaly Detection and Efficient Feature Extraction in IoT Networks. *IoT*, 4(3), 345-365.

[8] Luo, G., Xie, W., Gao, R., Zheng, T., Chen, L., & Sun, H. (2023). Unsupervised anomaly detection in brain MRI: Learning abstract distribution from massive healthy brains. *Computers in Biology and Medicine*, 154, 106610.

[9] Zhang, R., Isola, P., & Efros, A. A. (2016). Colorful image colorization. In *European Conference on Computer Vision (ECCV)* (pp. 649-666).

VAE: **References**

References: VAE

Reference Slides

- Fei-Fei Li, “Generative Deep Learning” (CS231)
- Hao Dong, “Deep Generative Models”
- Hung-Yi Lee, “Machine Learning”
- Murtaza Taj, “Deep Learning” (CS437)
- Aykut Erdem, COMP547: Deep Unsupervised Learning, Koc University
- P. Abbeel, P. Chen, J. Ho, A. Srinivas, A. Li, W. Yan, CS294-158: Deep Unsupervised Learning, UC Berkeley (Spring 2020), L4 Latent Variable Models
- A. van den Oord, “Neural Discrete Representation Learning,” SANE 2017 talk slides (DeepMind)
- Fei-Fei Li, J. Johnson, S. Yeung, CS231n Lecture 13: Generative Models, Stanford (2017)

References: VAE (cont.)

[1] D. P. Kingma and M. Welling,

"Auto-Encoding Variational Bayes,"

arXiv preprint arXiv:1312.6114, 2013.

[2] D. J. Rezende, S. Mohamed, and D. Wierstra,

"Stochastic Backpropagation and Approximate Inference in Deep Generative Models,"

Proceedings of the 31st International Conference on Machine Learning (ICML), 2014.

[3] I. Higgins, L. Matthey, A. Pal, et al.,

"beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework,"

International Conference on Learning Representations (ICLR), 2017.

References: VAE (cont.)

[4] C. P. Burgess, I. Higgins, A. Pal, et al.,
"Understanding Disentangling in β-VAE,"
arXiv preprint arXiv:1804.03599, 2018.

[5] K. Sohn, H. Lee, and X. Yan,
"Learning Structured Output Representation using Deep Conditional Generative Models,"
Advances in Neural Information Processing Systems (NeurIPS), 2015.

[6] E. Jang, S. Gu, and B. Poole,
"Categorical Reparameterization with Gumbel-Softmax,"
International Conference on Learning Representations (ICLR), 2017.

References: VAE (cont.)

[7] D. P. Kingma, T. Salimans, R. Jozefowicz, et al.,
"Improved Variational Inference with Inverse Autoregressive Flow,"
Advances in Neural Information Processing Systems (NeurIPS), 2016.

[8] J. Johnson, A. Alahi, and L. Fei-Fei,
"Perceptual Losses for Real-Time Style Transfer and Super-Resolution,"
European Conference on Computer Vision (ECCV), 2016.

[9] X. Hou, L. Shen, K. Sun, and G. Qiu,
"Deep Feature Consistent Variational Autoencoder,"
IEEE Winter Conference on Applications of Computer Vision (WACV), 2017.

References: VAE (cont.)

[10] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang,
"The Unreasonable Effectiveness of Deep Features as a Perceptual Metric (LPIPS),"
IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[11] A. van den Oord, O. Vinyals, and K. Kavukcuoglu,
"Neural Discrete Representation Learning,"
Advances in Neural Information Processing Systems (NeurIPS), 2017.

[12] A. van den Oord, N. Kalchbrenner, O. Vinyals, et al.,
"Conditional Image Generation with PixelCNN Decoders,"
Advances in Neural Information Processing Systems (NeurIPS), 2016.

References: VAE (cont.)

[13] A. van den Oord, S. Dieleman, H. Zen, et al.,

"WaveNet: A Generative Model for Raw Audio,"

arXiv preprint arXiv:1609.03499, 2016.

[14] A. Razavi, A. van den Oord, and O. Vinyals,

"Generating Diverse High-Fidelity Images with VQ-VAE-2,"

Advances in Neural Information Processing Systems (NeurIPS), 2019.

[15] P. Esser, R. Rombach, and B. Ommer,

"Taming Transformers for High-Resolution Image Synthesis (VQGAN),"

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

References: VAE (cont.)

[16] I. Higgins, D. Amos, D. Pfau, et al.,

"Towards a Definition of Disentangled Representations,"

arXiv preprint arXiv:1812.02230, 2018.

[17] C. Doersch,

"Tutorial on Variational Autoencoders,"

arXiv preprint arXiv:1606.05908, 2016.

[18] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe,

"Variational Inference: A Review for Statisticians,"

Journal of the American Statistical Association, 2017.