From Autoencoder to VAE: Reconstructing High-Dimensional Data with Neural Network Models

Presented by: Yue Yu†

†: Statistics Club, Indiana University

Tentative Schedule of Diffusion Model Series

- **10/10**: Overview of generative AI models in Computer Vision, including GANs (Goodfellow et al., 2014), VAEs (Kingma, 2013) and diffusion models (Sohl-Dickstein et al., 2015), and detailed introduction to GANs;
- **10/17: Detailed introduction to Autoencoders;**
- **10/24**: DDPM (Ho et al., 2020), the cornerstone paper about diffusion models, which enables diffusion models to produce high-quality, realistic samples and to be competitive with generative models like GANs and VAEs;

Tentative Schedule of Diffusion Model Series Cont'

- **11/7**: Score-based generative modeling (Song et al., 2020b), which bridges the gap between DDPMs and score-matching, offering a more generalized framework for diffusion-based generative models; and DDIM (Song et al., 2020a), which presents a method to speed up the sampling process in diffusion models without sacrificing much in terms of sample quality;

- **11/21**: Recent years' improvements and applications of diffusion models from different aspects, e.g., Latent Diffusion Models (Stable Diffusion, Rombach et al. (2022)) and its subsequent works like conditional control to text-to-image diffusion (Zhang et al., 2023) and Stable Diffusion XL (Podell et al., 2023).

Today's Presentation Outline

- Autoencoders (AEs);
- Variational Autoencoders (VAEs) (Kingma, 2013), and ELBO;
- Comparison Between Different Models (GANs, VAEs, Diffusion Models).

Notation (1/2)

- $\mathcal{D}$: The dataset, $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\}$, contains n data samples; $|\mathcal{D}| = n$.
- $\mathbf{x}^{(i)}$: Each data point is a vector of d dimensions, $\mathbf{x}^{(i)} = \left[x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}\right]$.
- $\mathbf{x}$: One data sample from the dataset, $\mathbf{x} \in \mathcal{D}$.
- $\mathbf{x}'$: The reconstructed version of $\mathbf{x}$ (generated data).
- $\tilde{\mathbf{x}}$: The corrupted version of $\mathbf{x}$.
- $\mathbf{z}$: The compressed code learned in the bottleneck layer.

Notation (2/2)

- $a_j^{(l)}$: The activation function for the j-th neuron in the l-th hidden layer.
- $g_\phi(\cdot)$: The encoding function parameterized by ϕ.
- $f_\theta(\cdot)$: The decoding function parameterized by θ.
- $q_\phi(\mathbf{z} \mid \mathbf{x})$: Estimated posterior probability function, also known as probabilistic encoder.
- $p_\theta(\mathbf{x} \mid \mathbf{z})$: Likelihood of generating true data sample given the latent code, also known as probabilistic decoder.

Autoencoder

Autoencoder is a neural network designed to learn an identity function in an unsupervised way to reconstruct the original input while compressing the data in the process so as to discover a more efficient and compressed representation. The idea originated in the 1980s (Oja, 1982), and was later promoted by Hinton and Salakhutdinov (2006).

It consists of two networks:

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

Autoencoder Architecture

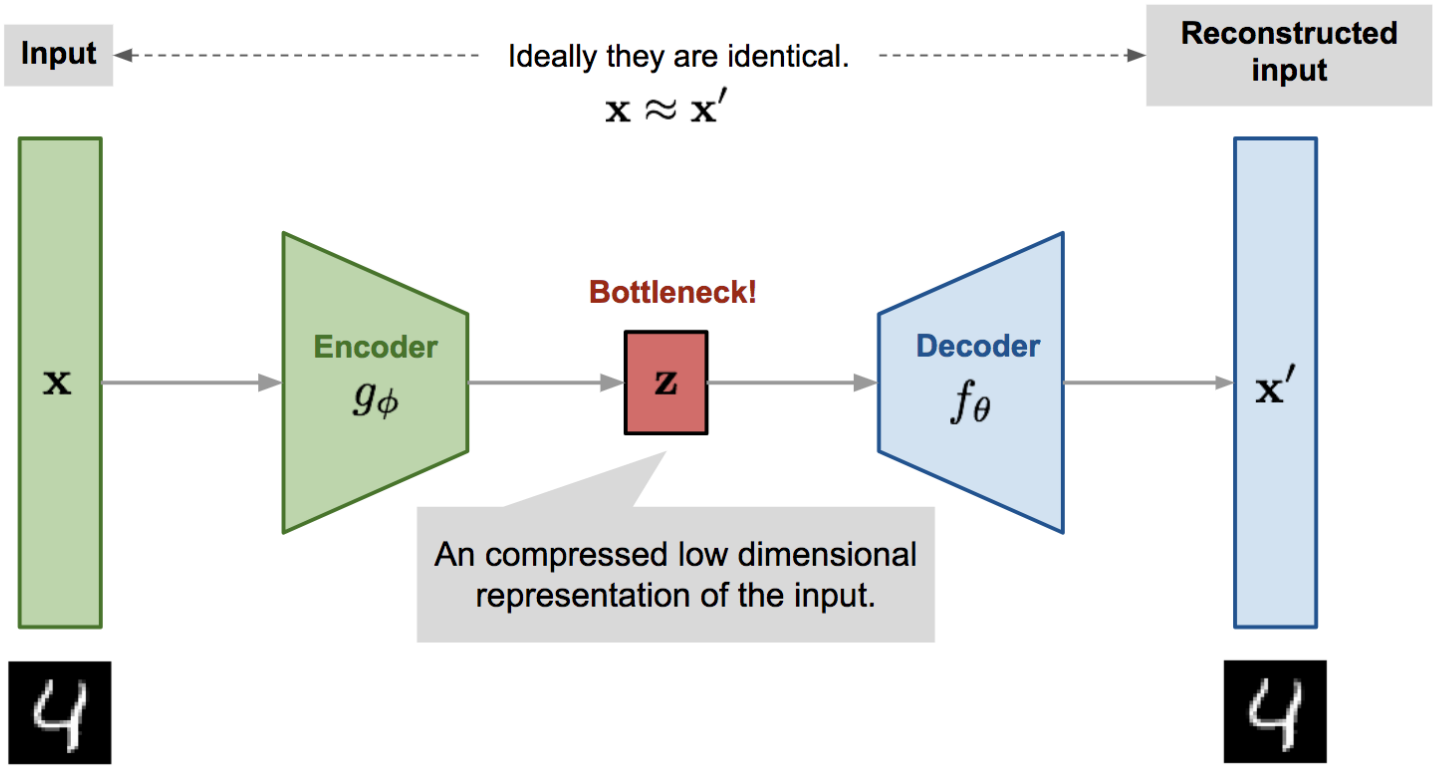


Figure: Illustration of autoencoder model architecture. The encoder network essentially accomplishes the **dimensionality reduction**, just like how we would use Principal Component Analysis (PCA) for.

Autoencoder Details

The model contains an encoder function $g(\cdot)$ parameterized by ϕ and a decoder function $f(\cdot)$ parameterized by θ. The low-dimensional code learned for input $\mathbf{x}$ in the bottleneck layer is $\mathbf{z} = g_\phi(\mathbf{x})$ 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 original input, $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$, or in other words, to learn an identity function. There are various metrics to quantify the difference between two vectors, such as cross entropy when the activation function is sigmoid, or as simple as MSE loss:

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

Denoising Autoencoder

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

To avoid overfitting and improve the robustness, **Denoising Autoencoder** (Vincent et al., 2008) proposed a modification to the basic autoencoder. The input is partially corrupted by adding noise or masking some values of the input vector in a stochastic manner, $\tilde{\mathbf{x}} \sim \mathcal{M}_{\mathcal{D}}(\tilde{\mathbf{x}} \mid \mathbf{x})$. Then the model is trained to recover the original input (not the corrupt one).

$$\tilde{\mathbf{x}}^{(i)} \sim \mathcal{M}_{\mathcal{D}}(\tilde{\mathbf{x}}^{(i)} \mid \mathbf{x}^{(i)})$$

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

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

Denoising Autoencoder Architecture

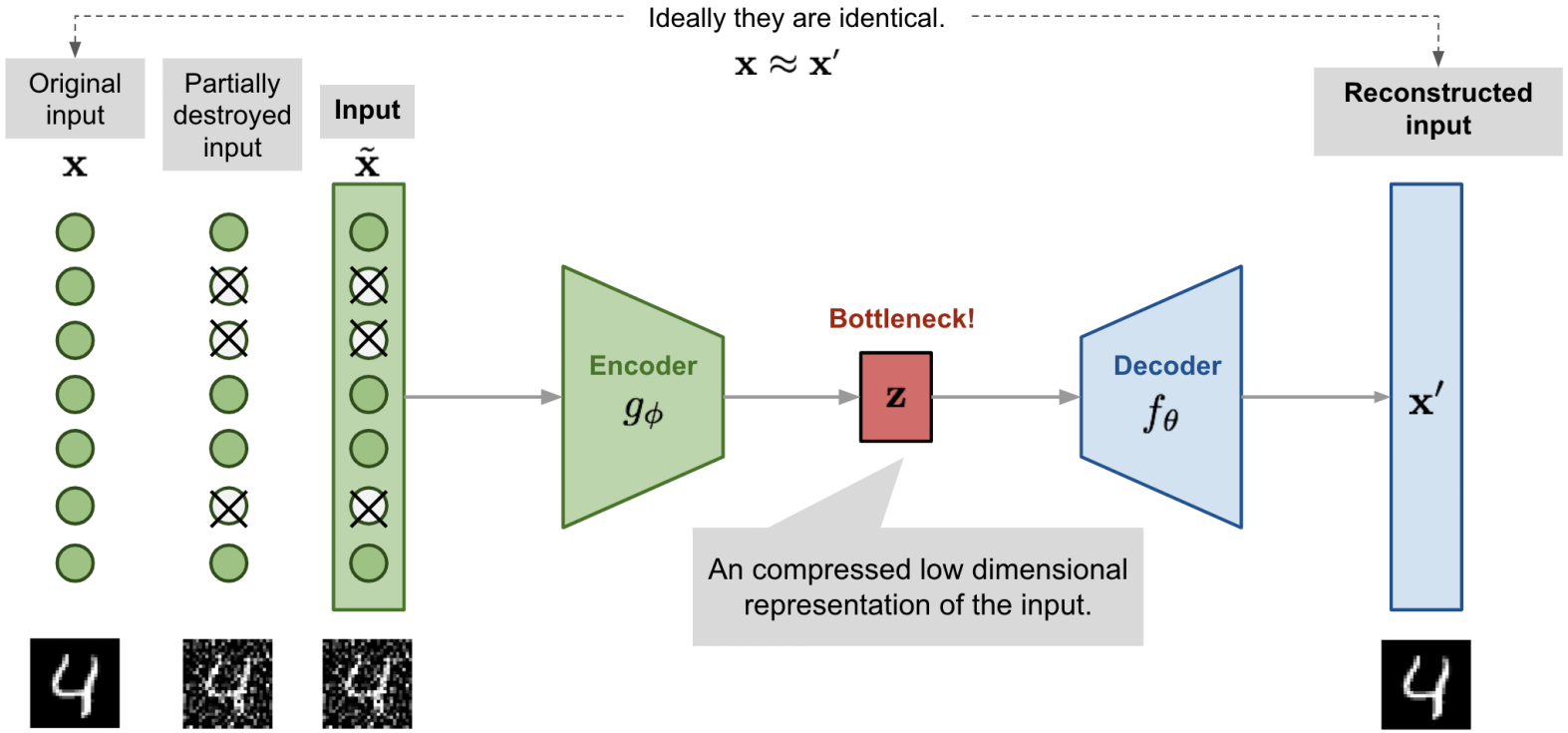


Figure: Illustration of denoising autoencoder model architecture. This design is motivated by the fact that humans can easily recognize an object or a scene even the view is partially occluded or corrupted. To "repair" the partially destroyed input, the denoising autoencoder has to discover and capture relationship between dimensions of input in order to infer missing pieces.

Corruption Process in Denoising Autoencoder

- For high-dimensional input with high redundancy, like images, the model is likely to depend on evidence gathered from a combination of many input dimensions to recover the denoised version rather than to overfit one dimension. This builds up a good foundation for learning *robust* latent representation.
- The noise is controlled by a stochastic mapping $\mathcal{M}_{\mathcal{D}}(\tilde{\mathbf{x}} \mid \mathbf{x})$, and it is not specific to a particular type of corruption process (i.e., Gaussian noise). Naturally, the corruption process can be equipped with prior knowledge. It's also similar to the data augmentation process, which is a commonly used technique to improve the robustness of NNs.
- In the experiment of the original DAE paper, the noise is applied in this way: a fixed proportion of input dimensions are selected at random and their values are forced to 0. Sounds a lot like dropout, but that idea was actually proposed earlier (Srivastava et al., 2014).

Sparse Autoencoder

Sparse Autoencoder (Ng et al., 2011) adds a constraint to limit the number of active hidden units, reducing overfitting and improving robustness. Common activation functions (e.g., sigmoid, tanh, ReLU)

activate neurons near 1 and deactivate them near 0. In the l-th hidden layer with s_l neurons, the activation of neuron j, $a_j^{(l)}(\cdot)$, is averaged over n samples to match a target sparsity ρ, typically set to 0.25:

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

Sparsity Constraint in Sparse Autoencoder

The sparsity constraint is added as a penalty term in the loss function using KL-divergence, D_{KL}, which measures the difference between two Bernoulli distributions with means ρ and $\hat{\rho}_j^{(l)}$. The hyperparameter β controls the penalty strength.

$$L_{\mathsf{SAE}}(\theta) = L(\theta) + \beta \sum_{l=1}^{L} \sum_{j=1}^{s_l} D_{\mathsf{KL}}(\rho \parallel \hat{\rho}_j^{(l)})$$

$$= L(\theta) + \beta \sum_{l=1}^{L} \sum_{j=1}^{s_l} \left[\rho \log \frac{\rho}{\hat{\rho}_j^{(l)}} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j^{(l)}} \right]$$

KL Divergence for Sparsity Constraint

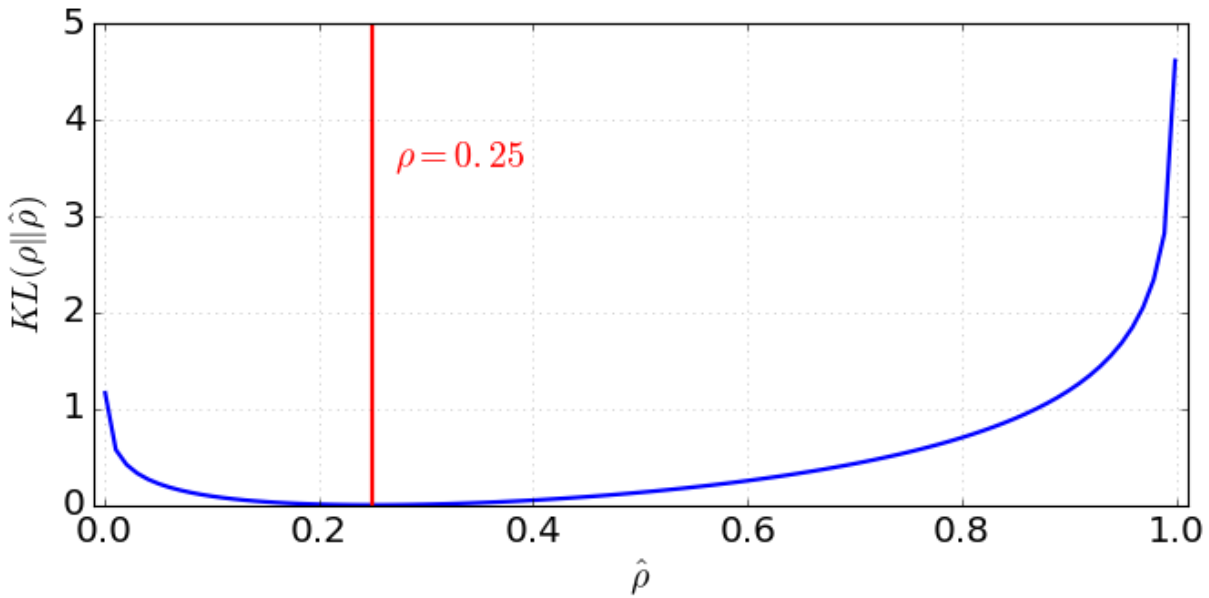


Figure: The KL divergence between a Bernoulli distribution with mean $\rho = 0.25$ and a Bernoulli distribution with mean $0 \leq \hat{\rho} \leq 1$.

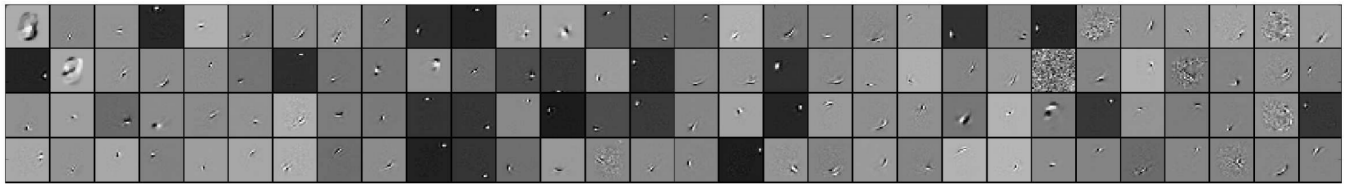
k-Sparse Autoencoder

In a k-**Sparse Autoencoder** (Makhzani and Frey, 2013), sparsity is enforced by retaining the top k highest activations in the bottleneck layer.

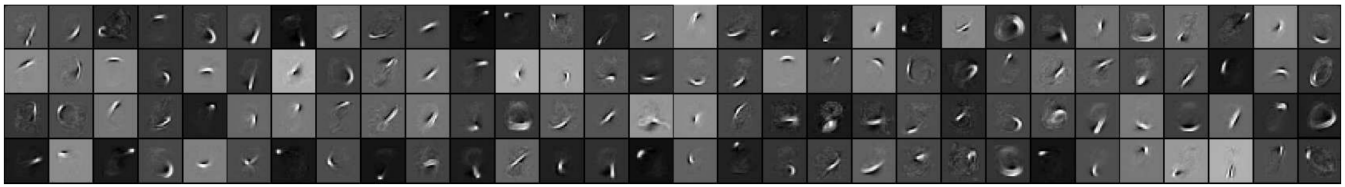
- Run the encoder to get the compressed code: $\mathbf{z} = g(\mathbf{x})$.
- Sort $\mathbf{z}$ and keep only the top k largest values, setting others to 0: $\mathbf{z}' = \text{Sparsify}(\mathbf{z})$.
- Optionally, use a ReLU layer with an adjustable threshold.
- Compute the output and loss: $L = \|\mathbf{x} - f(\mathbf{z}')\|_2^2$.

Backpropagation is limited to the top k activated units.

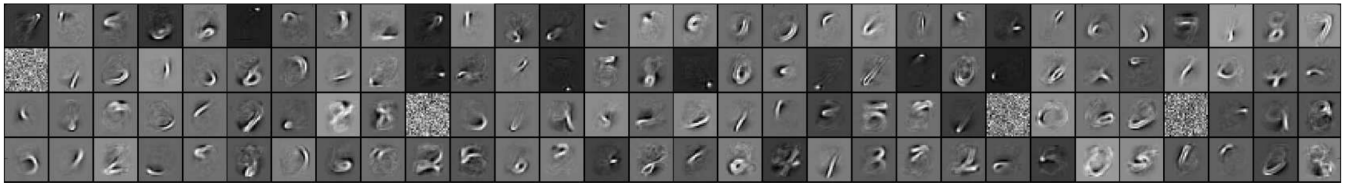
k-Sparse Autoencoder Filters



(a) $k = 70$



(b) $k = 40$



(c) $k = 25$

(d) $k = 10$

Figure: Filters of the k-Sparse Autoencoder for different sparsity levels k, learned from MNIST with 1000 hidden units.

VAE: Variational Autoencoder

The concept of a **Variational Autoencoder (VAE)** (Kingma, 2013), is distinct from traditional autoencoders. It is rooted in variational Bayesian methods and graphical models.

Instead of mapping the input to a fixed vector, the VAE maps it to a distribution, denoted as p_θ, parameterized by θ. The relationship between the input $\mathbf{x}$ and the latent encoding $\mathbf{z}$ involves:

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

Generating Samples with VAE

To generate a sample that resembles a real data point $\mathbf{x}^{(i)}$, assuming we know the true parameter θ^*, follow these steps:

1. Sample $\mathbf{z}^{(i)}$ from the prior distribution $p_{\theta^*}(\mathbf{z})$.
2. Generate $\mathbf{x}^{(i)}$ from the conditional distribution $p_{\theta^*}(\mathbf{x} \mid \mathbf{z} = \mathbf{z}^{(i)})$.

The optimal parameter θ^* maximizes the probability of the observed data:

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

Using the log probability, we convert the product to a sum:

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

Updating the Data Generation Process in VAE

To involve the encoding vector, we update the equation for the data generation process:

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

Computing $p_\theta(\mathbf{x}^{(i)})$ directly is difficult due to the need to sum over all possible values of $\mathbf{z}$. To simplify this, we introduce an approximation function $q_\phi(\mathbf{z} \mid \mathbf{x})$, parameterized by ϕ, to narrow the value space and facilitate faster computation.

Graphical Model of Variational Autoencoder

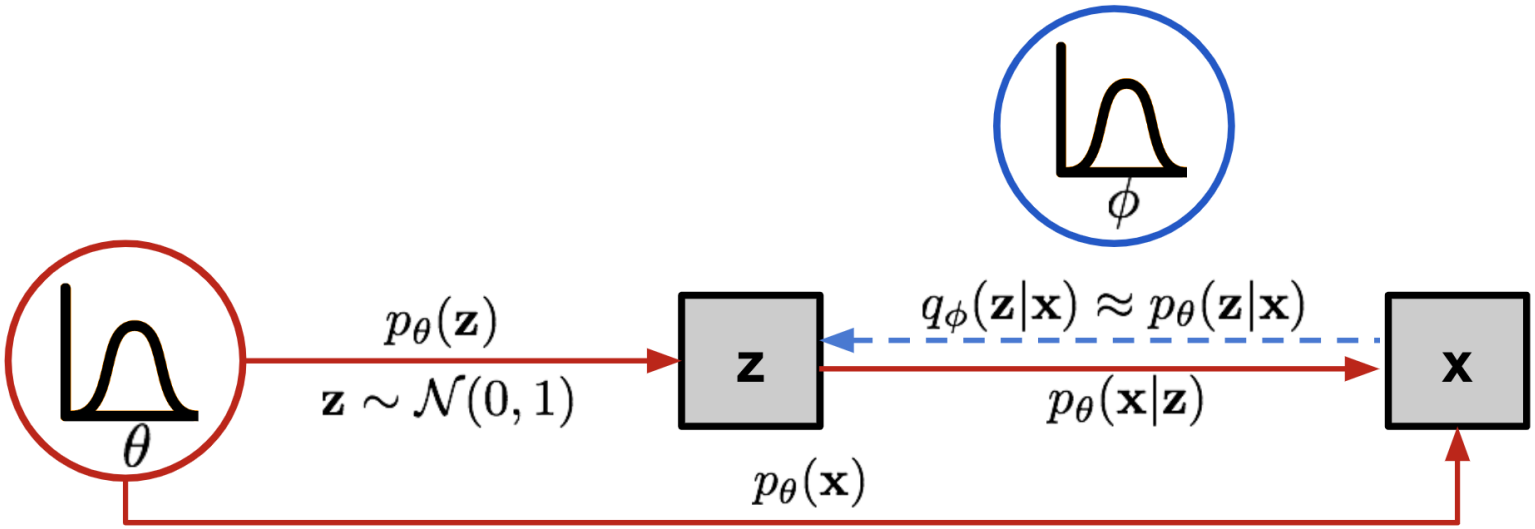


Figure: The graphical model involved in Variational Autoencoder. Solid lines denote the generative distribution $p_\theta(\cdot)$, and dashed lines denote the distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ to approximate the intractable posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$.

VAE Structure and Autoencoder Similarity

Now, the structure resembles an autoencoder:

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

Minimizing KL Divergence in VAE

The estimated posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ should closely approximate the true posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$. We use **Kullback-Leibler divergence** to measure the distance between these two distributions.

The KL divergence, $D_{\mathsf{KL}}(X \| Y)$, quantifies how much information is lost when using distribution Y to approximate X.

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

Expanding the KL Divergence Equation (Part 1)

Let's expand the equation:

$$
\begin{aligned}
D_{\mathsf{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z} \mid \mathbf{x})) &= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z} \mid \mathbf{x}) \left(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) d\mathbf{z} \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z} \mid \mathbf{x}) \log \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z}
\end{aligned}
$$

Expanding the KL Divergence Equation (Part 2)

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

Rearranging the KL Divergence Equation

Once we rearrange the equation, we obtain:

$$\begin{aligned}&\log p_\theta(\mathbf{x}) - D_{\mathsf{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z} \mid \mathbf{x})) \\ =&\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] - D_{\mathsf{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z}))\end{aligned}$$

The left-hand side (LHS) represents our objective: maximizing the (log-)likelihood of generating real data, $\log p_\theta(\mathbf{x})$, while minimizing the difference between the true and estimated posterior distributions, with D_{KL} serving as a regularizer.

Note that $p_\theta(\mathbf{x})$ is fixed with respect to q_ϕ.

VAE Loss Function and ELBO

The negation of the previous equation defines the VAE loss function:

$$\begin{aligned} L_{\text{VAE}}(\theta, \phi) &= -\log p_\theta(\mathbf{x}) + D_{\text{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z})) \\ &= -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] + D_{\text{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z})) \end{aligned}$$

The optimal parameters θ^*, ϕ^* are obtained by minimizing L_{VAE}:

$$\theta^*, \phi^* = \arg\min_{\theta, \phi} L_{\text{VAE}}$$

In Variational Bayesian methods, this loss is known as **E**vidence **L**ower **Bo**und (ELBO):

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

Therefore, minimizing L_{VAE} maximizes the lower bound of the probability of generating real data samples.

Reparameterization Trick

- The expectation term in the loss function involves sampling from $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x})$, making backpropagation infeasible.
- To address this, the reparameterization trick expresses the random variable $\mathbf{z}$ as:

$$\mathbf{z} = \mathcal{T}_\phi(\mathbf{x}, \boldsymbol{\epsilon}), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$

  where $\boldsymbol{\epsilon}$ is an auxiliary independent random variable.
- A common choice for $q_\phi(\mathbf{z} \mid \mathbf{x})$ is a multivariate Gaussian:

$$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 I)$$
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

- Here, $\odot$ denotes element-wise multiplication.
- This noise $\boldsymbol{\epsilon}$ adds the necessary stochasticity while keeping the sampling process differentiable.

VAE Architecture

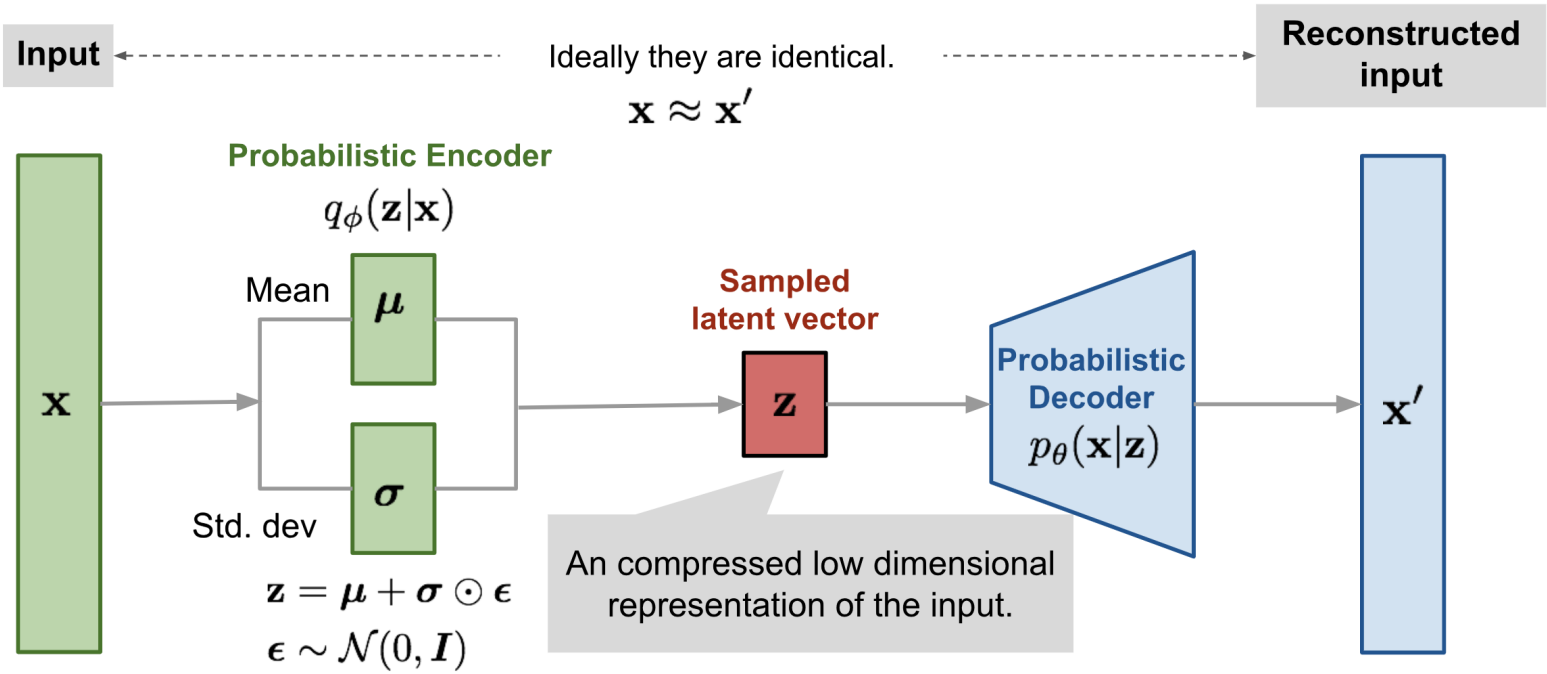


Figure: Illustration of variational autoencoder model with the multivariate Gaussian assumption.

Relationship between VAEs and GANs

- Both VAEs and GANs are generative models but use different approaches.
- **VAE**: Uses a probabilistic framework with a defined likelihood.
- **GAN**: Uses a discriminator to distinguish between real and generated samples, training the generator to fool the discriminator.
- VAEs focus on learning latent representations, while GANs aim to generate realistic samples.

Diffusion Models (Next Talk)

- Diffusion models generate data by simulating a gradual denoising process.
- Starts with a noisy version of data and iteratively removes noise using a learned denoising network.
- The training objective minimizes the difference between the denoised output and the original data.

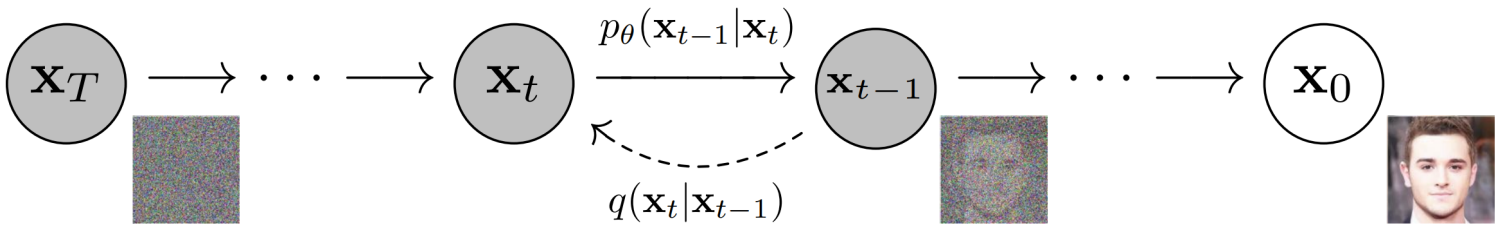
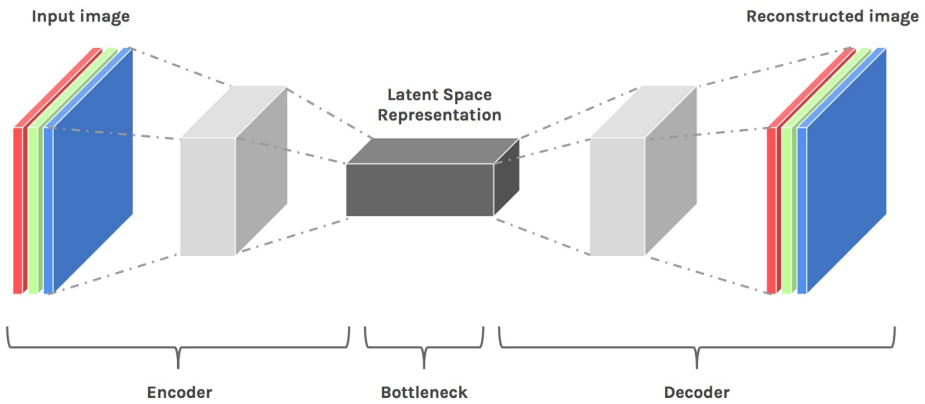


Figure: Illustration of diffusion model forward and reverse processes (Ho et al., 2020).

Stable Diffusion and State-of-the-Art

- **Stable Diffusion**: A type of latent diffusion model that operates in the latent space of an autoencoder (Rombach et al., 2022).
- Combines the power of VAEs for encoding data into a compressed latent space and diffusion models for denoising.
- Achieves state-of-the-art results in image synthesis with high fidelity and diversity.



https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df

Questions?

Thank you!

References I

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

D. P. Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

A. Makhzani and B. Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

A. Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

References II

E. Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15:267–273, 1982.

D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a.

References III

Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

L. Zhang, A. Rao, and M. Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.