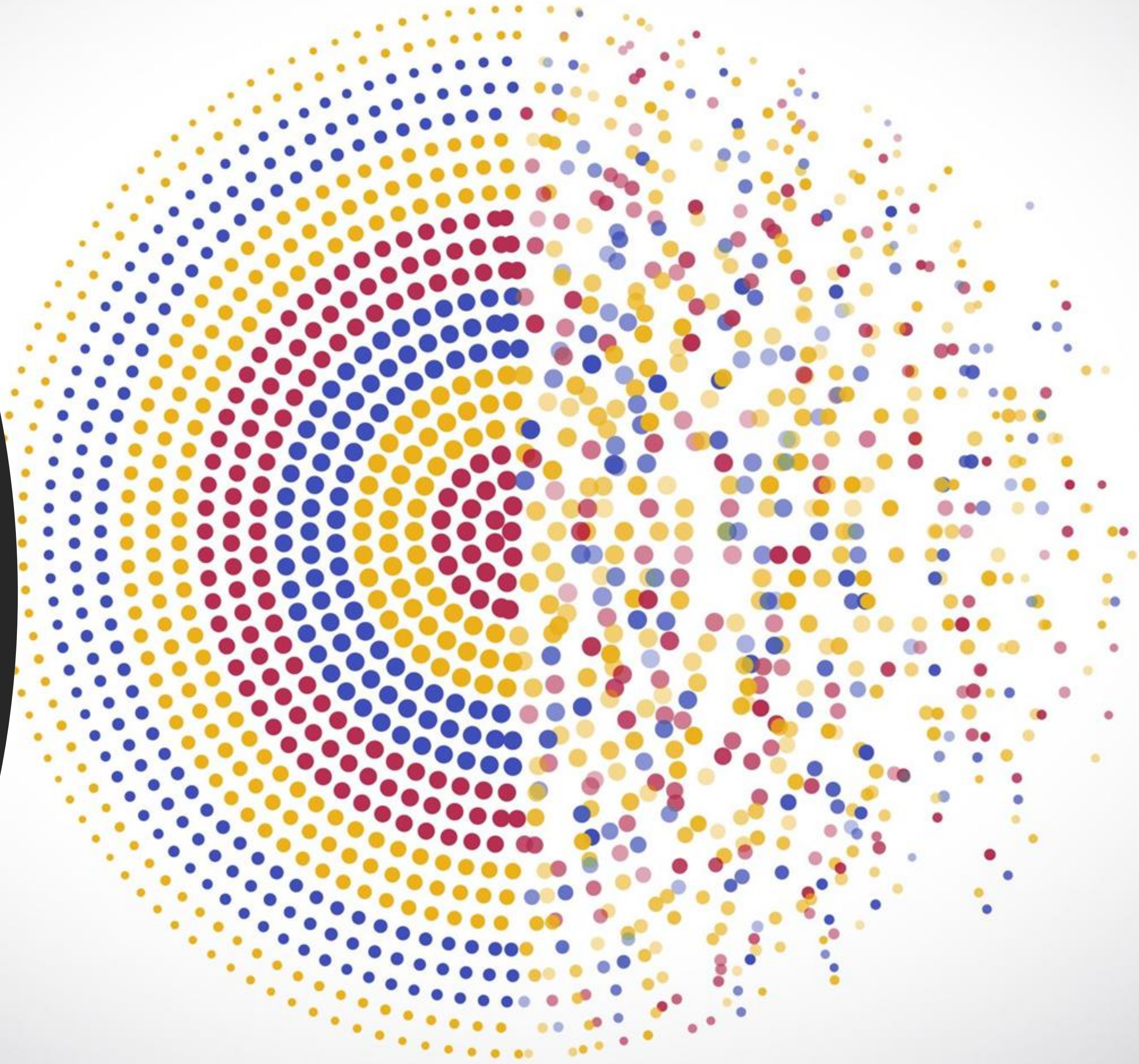
# Unsupervised Learning

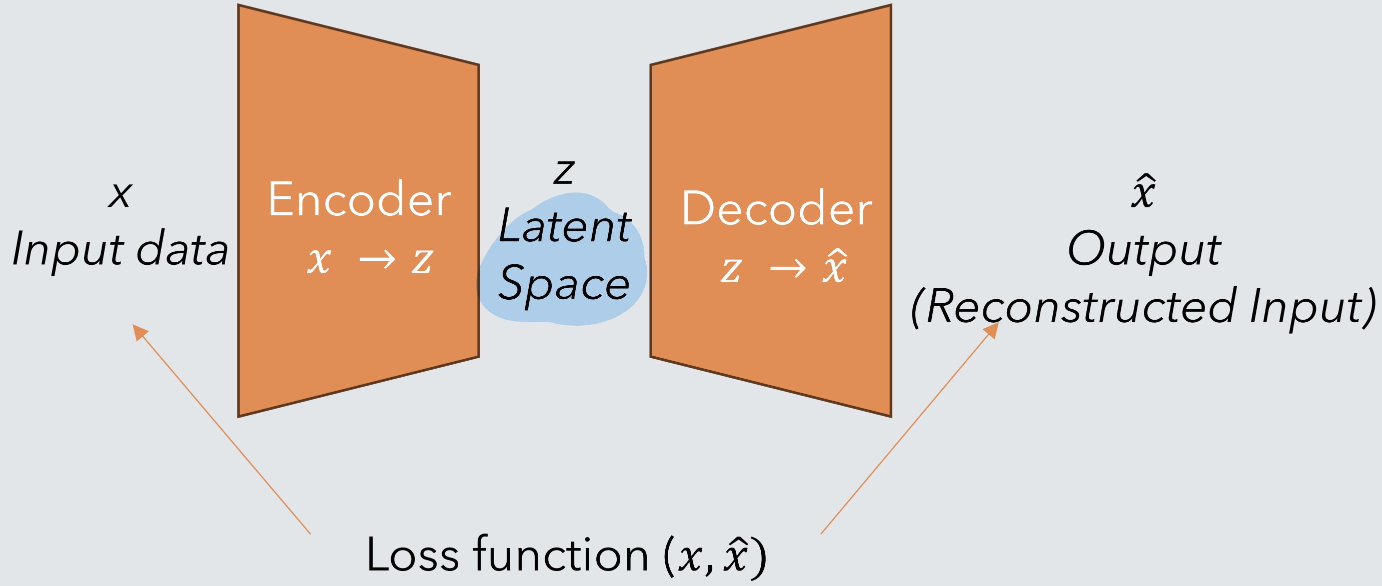
Dr. Mohamed AlHajri



# Content

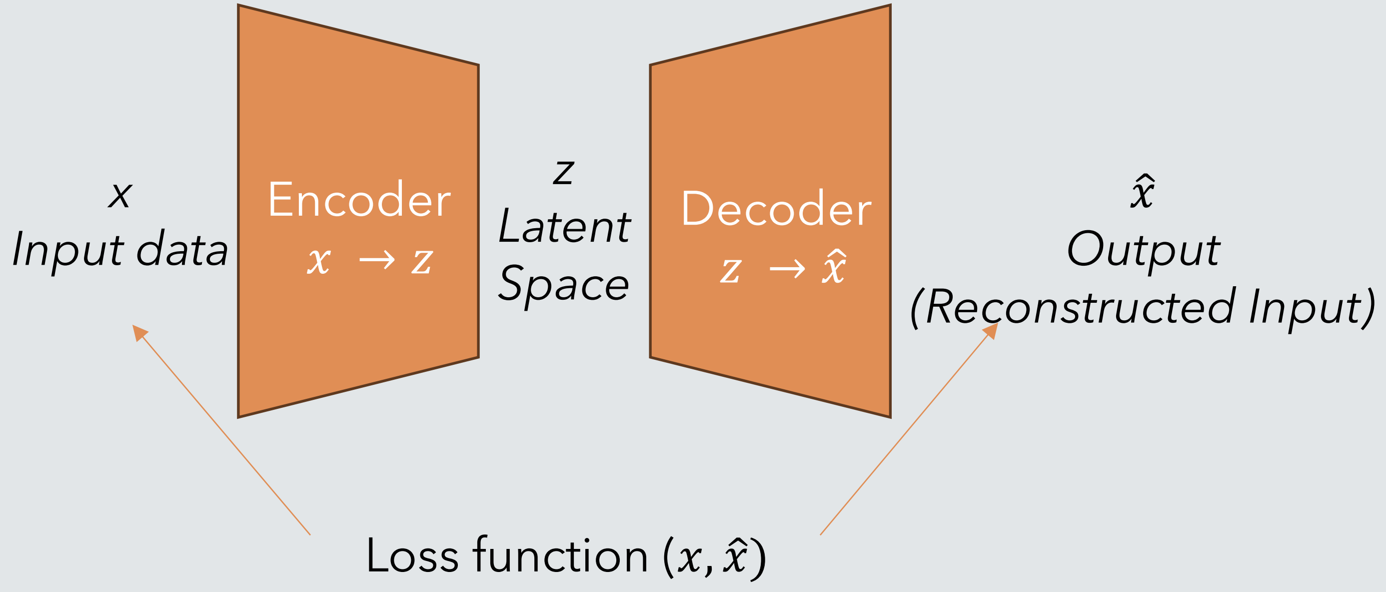
- *Vanilla Autoencoders*
- *Denoising Autoencoders*
- *Sparse Autoencoders*
- *Convolutional Autoencoder*
- *Variational Autoenocder*

# Autoencoder



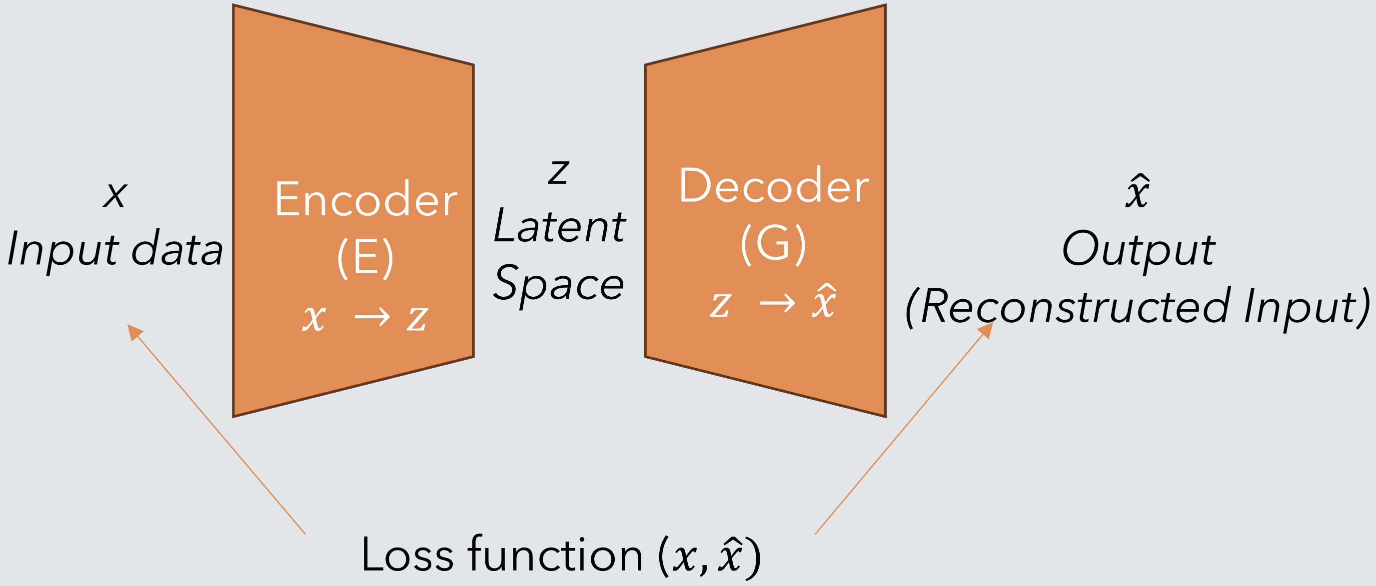

- Reconstruct high-dimensional data using a neural network model with a narrow bottleneck layer.
- The bottleneck layer captures the compressed latent coding, so the nice by-product is dimension reduction.
- The low-dimensional representation can be used as the representation of the data in various applications, e.g., image retrieval, data compression

# Autoencoder




- Encoder network will be a neural network where the dimensions of the output will usually be smaller than the input and it will be used for dimensionality reduction.
- If dim(z) < dim(x) [Undercomplete system]
- If dim(x) > dim(z) [Overcomplete system] (not popular since no compression)

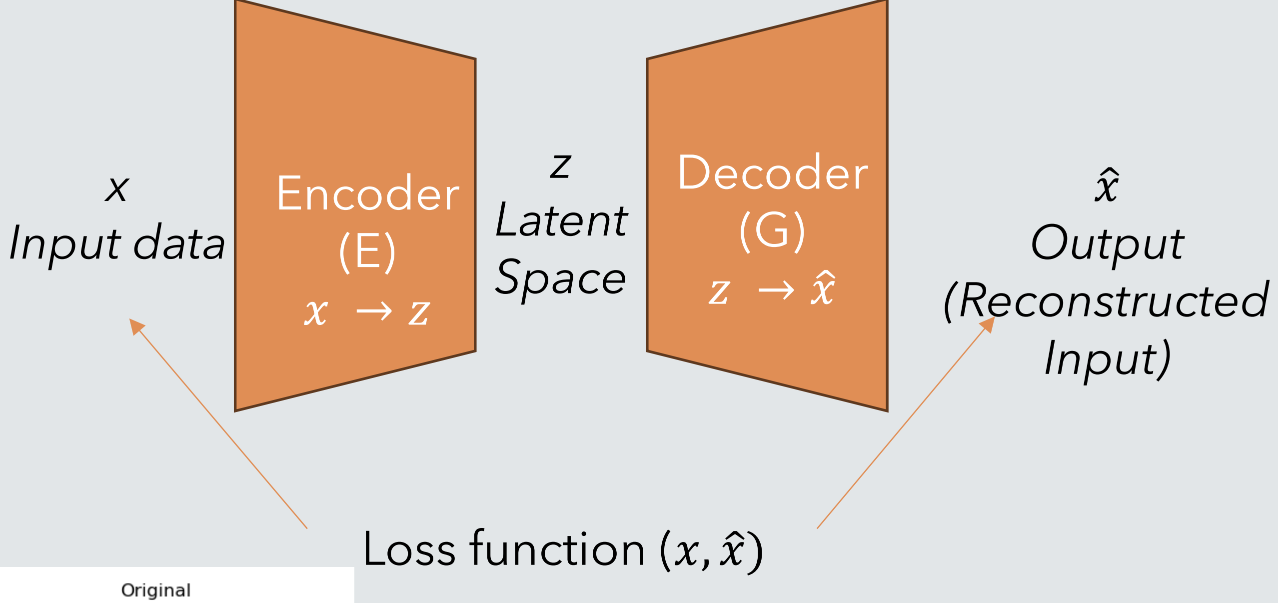
# Autoencoder

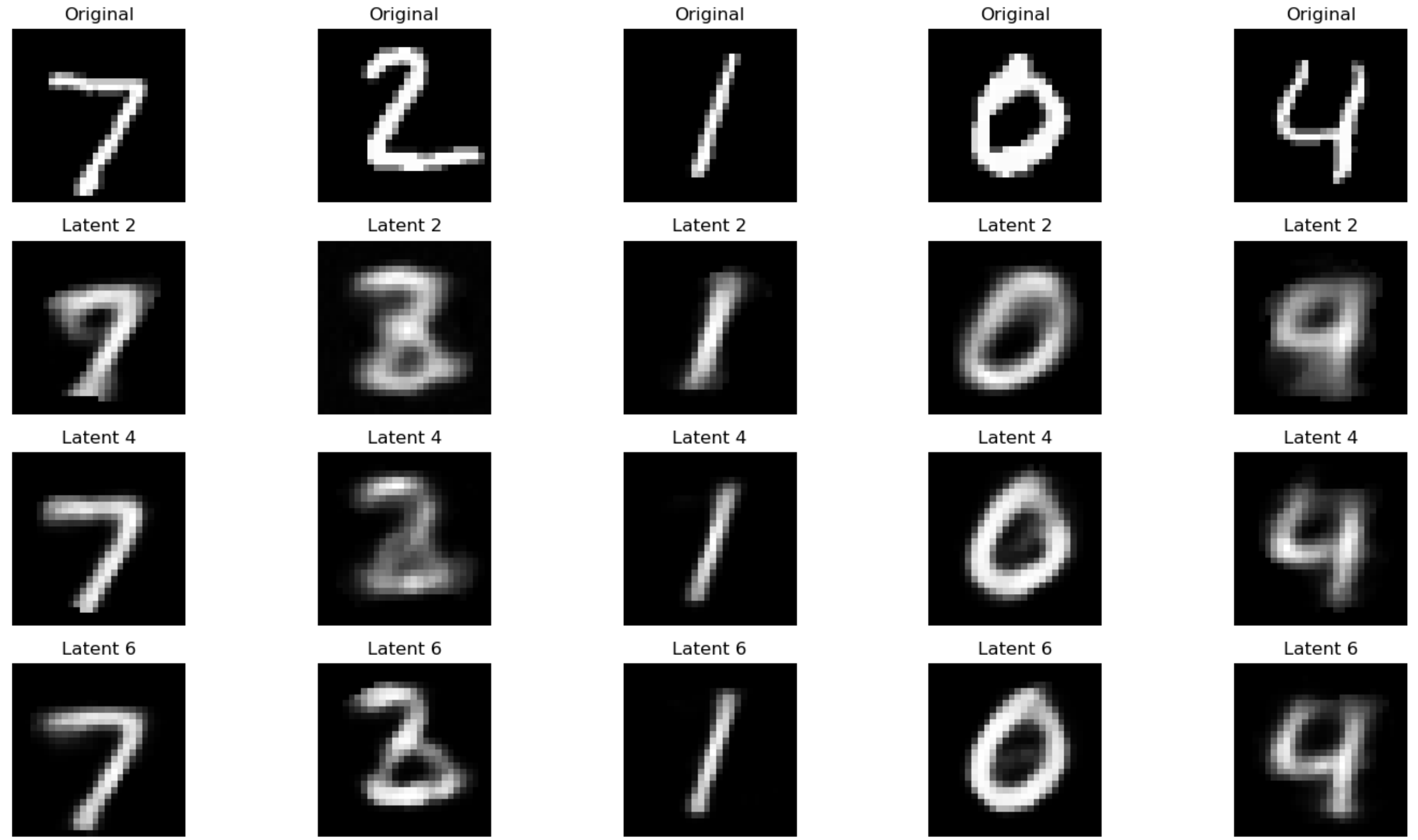



- The distance between the input and output data will be measured using the loss function:

$$\boxed{L = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - G\left(E(x_i)\right)\right)^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}$$

# Autoencoder

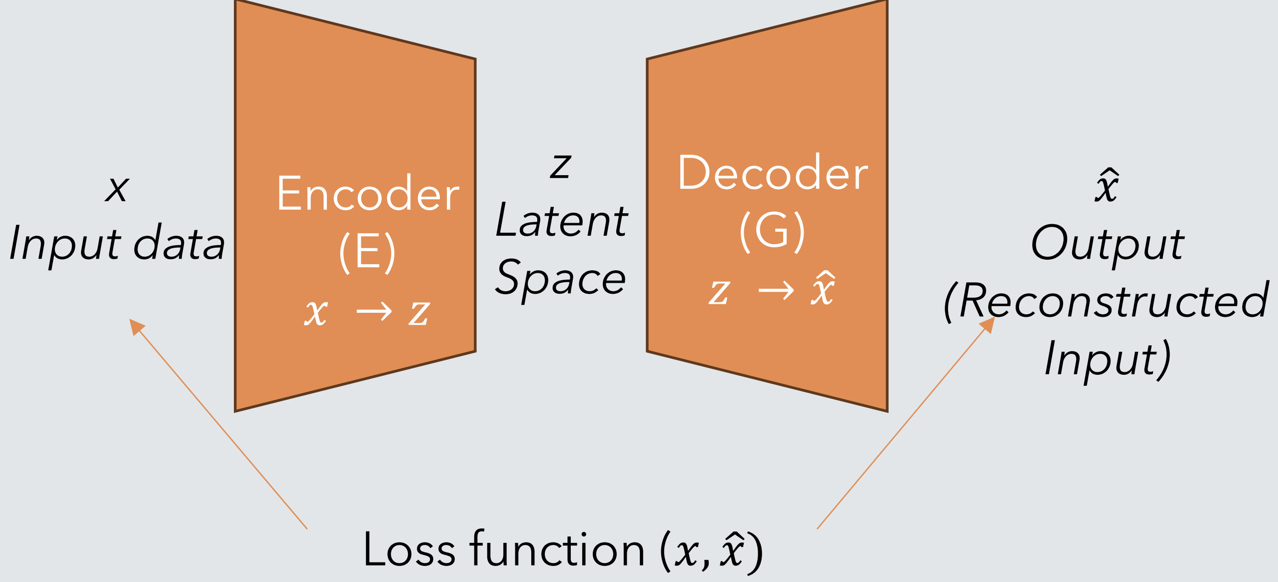


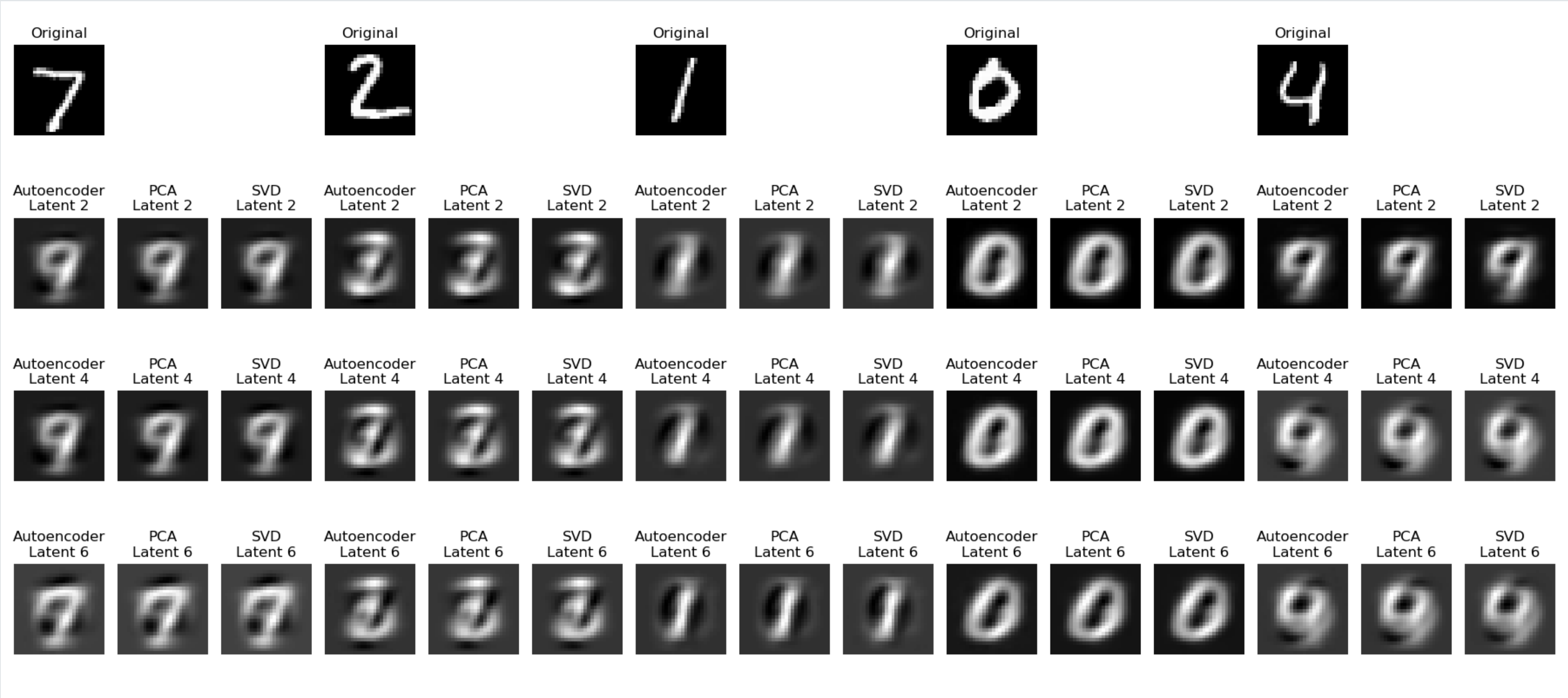




| Latent Dimension | Loss |
|---|---|
| 2 | 0.048 |
| 4 | 0.033 |
| 6 | 0.027 |

# Autoencoder

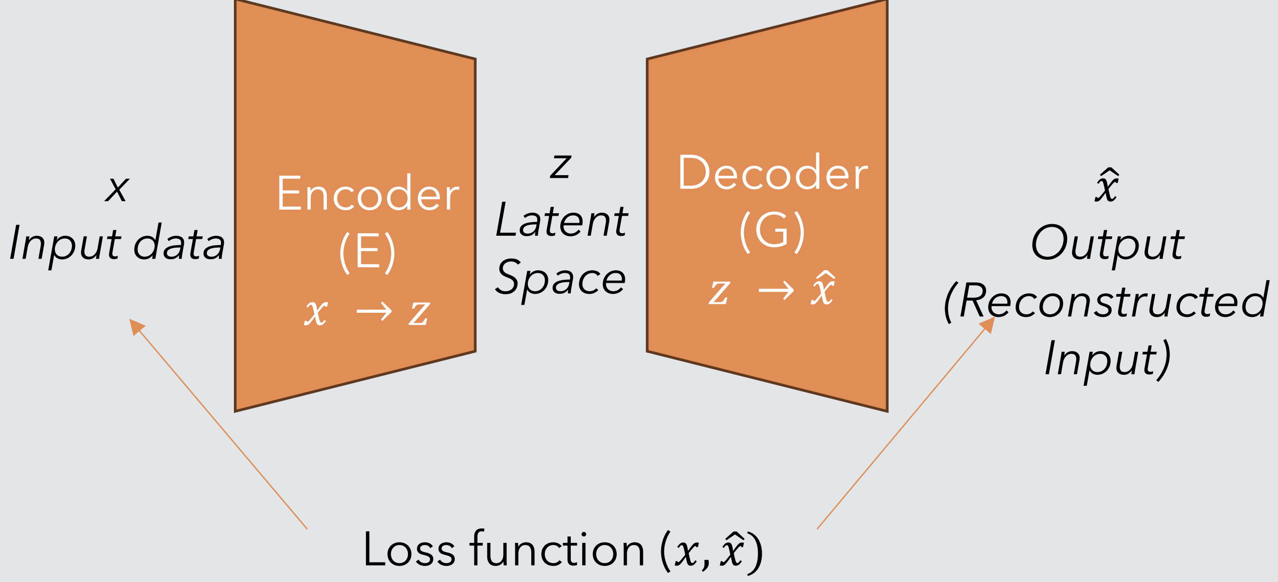
What happens if we have linear activation functions?

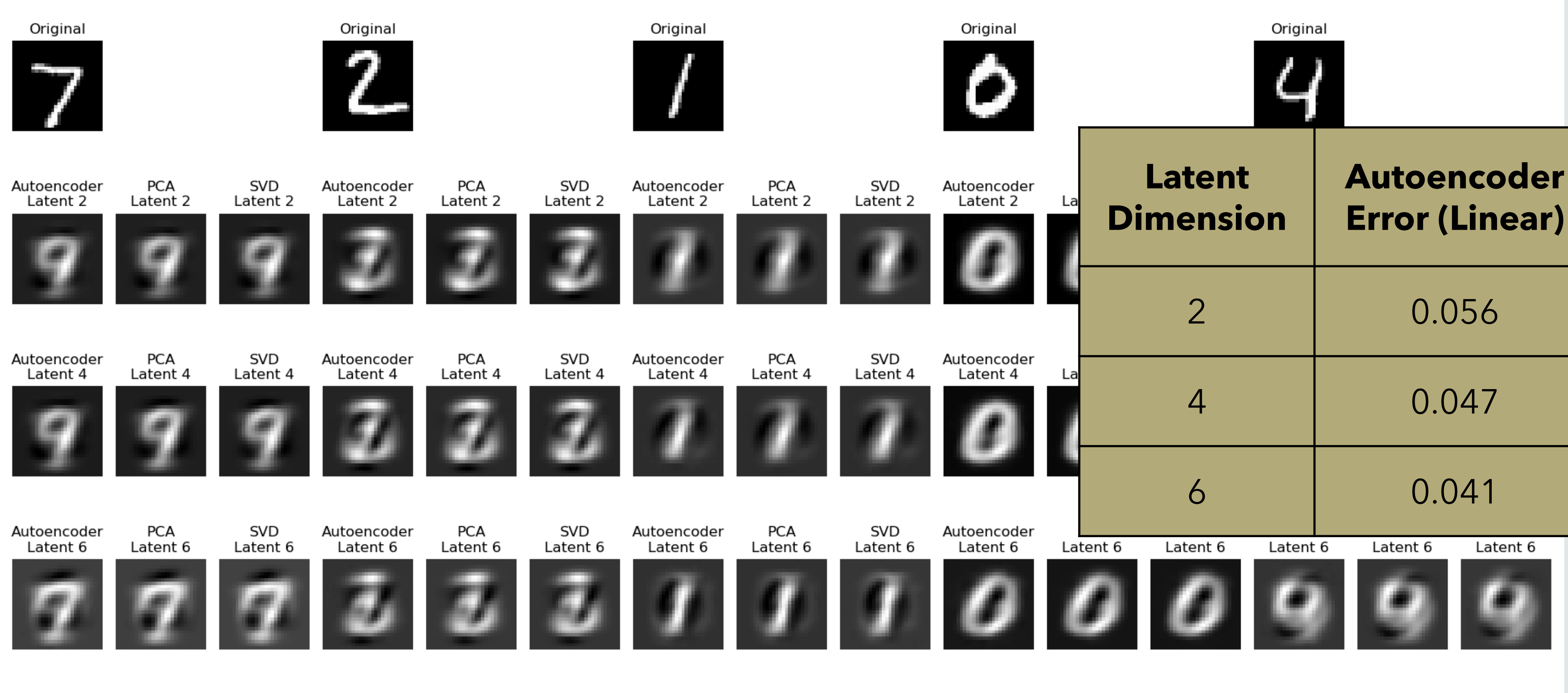


# Autoencoder

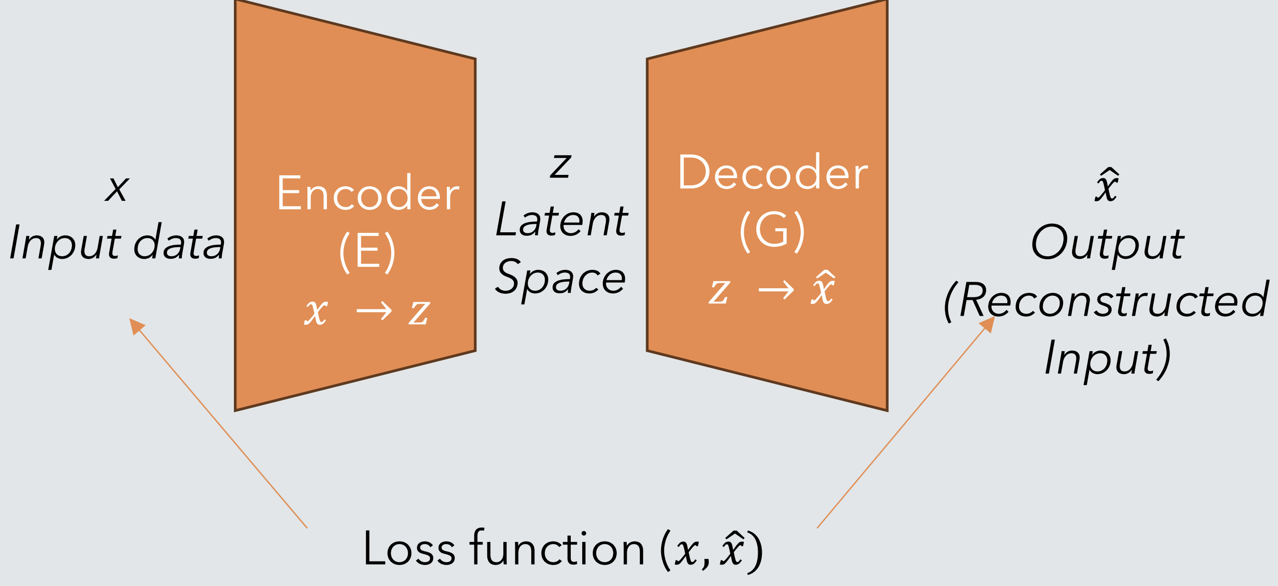
What happens if we have linear activation functions?




| Latent Dimension | Autoencoder Error (Linear) | PCA Error | SVD Error |
|---|---|---|---|
| 2 | 0.056 | 0.055 | 0.055 |
| 4 | 0.047 | 0.047 | 0.047 |
| 6 | 0.041 | 0.041 | 0.041 |

# Autoencoder

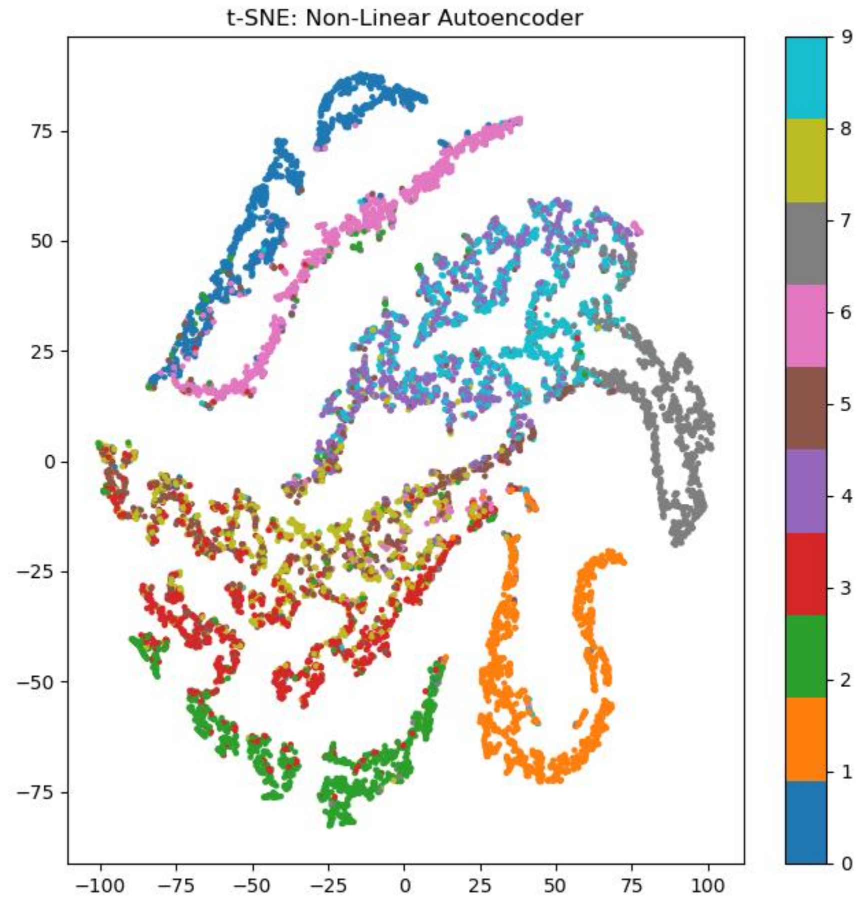


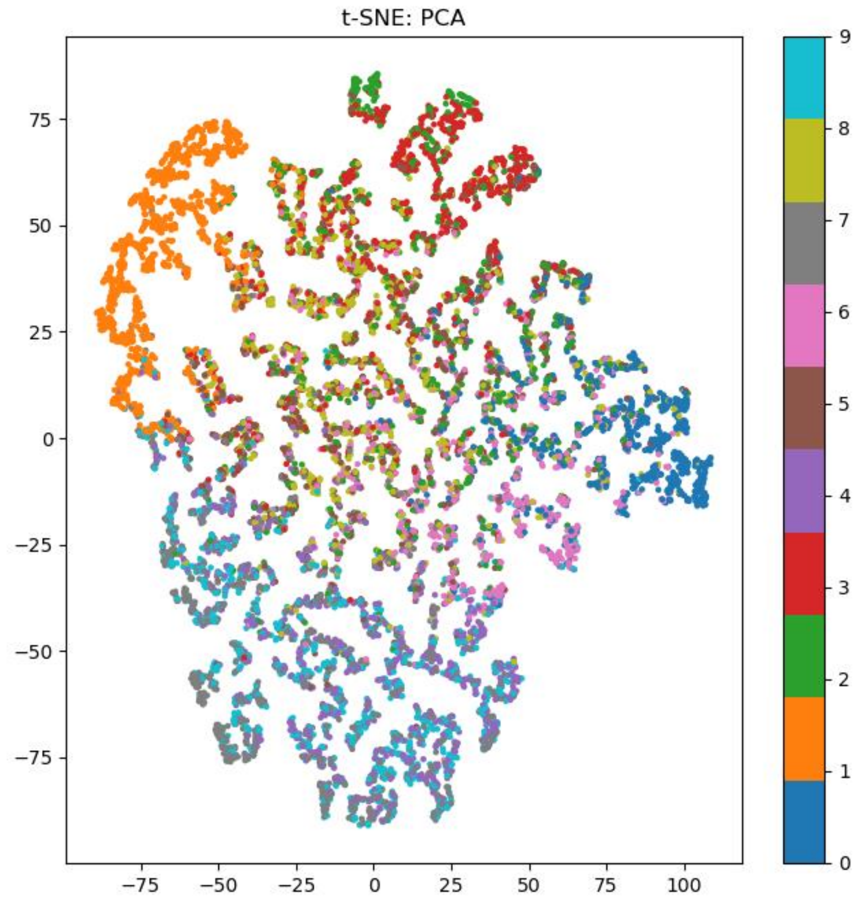

What happens if we have linear activation functions?

- Very small differences due to some difference in implementation.
- However, an autoencoder with a linear activation function does something very similar to pca and svd, which is simply a dimensionality reduction approach.
- We can think about it as minimizing mse, or maximizing variance.
- Another way to look at it is that it is low rank approximation.

| Latent Dimension | Autoencoder Error (Linear) | PCA Error | SVD Error |
|---|---|---|---|
| 2 | 0.056 | 0.055 | 0.055 |
| 4 | 0.047 | 0.047 | 0.047 |
| 6 | 0.041 | 0.041 | 0.041 |

# Autoencoder



# *Denoising Autoencoder*

- A **denoising autoencoder (DAE)** is a variant of the traditional autoencoder designed to improve robustness and avoid learning a trivial identity mapping by introducing noise into the input data. The model learns to map noisy inputs back to their clean, original counterparts. This forces the encoder to extract meaningful features and is particularly useful for **denoising** and **dimensionality reduction**.
- **Key Concepts of Denoising Autoencoder:**

  ***1. Noise Injection***:

  1. Add noise to the input (e.g., Gaussian noise, salt-and-pepper noise, masking noise).
  2. This noise forces the autoencoder to learn more robust latent representations rather than simply copying input to output.

# *Denoising Autoencoder*

- **Key Concepts of Denoising Autoencoder:**

**_2. Learning Objective_**:

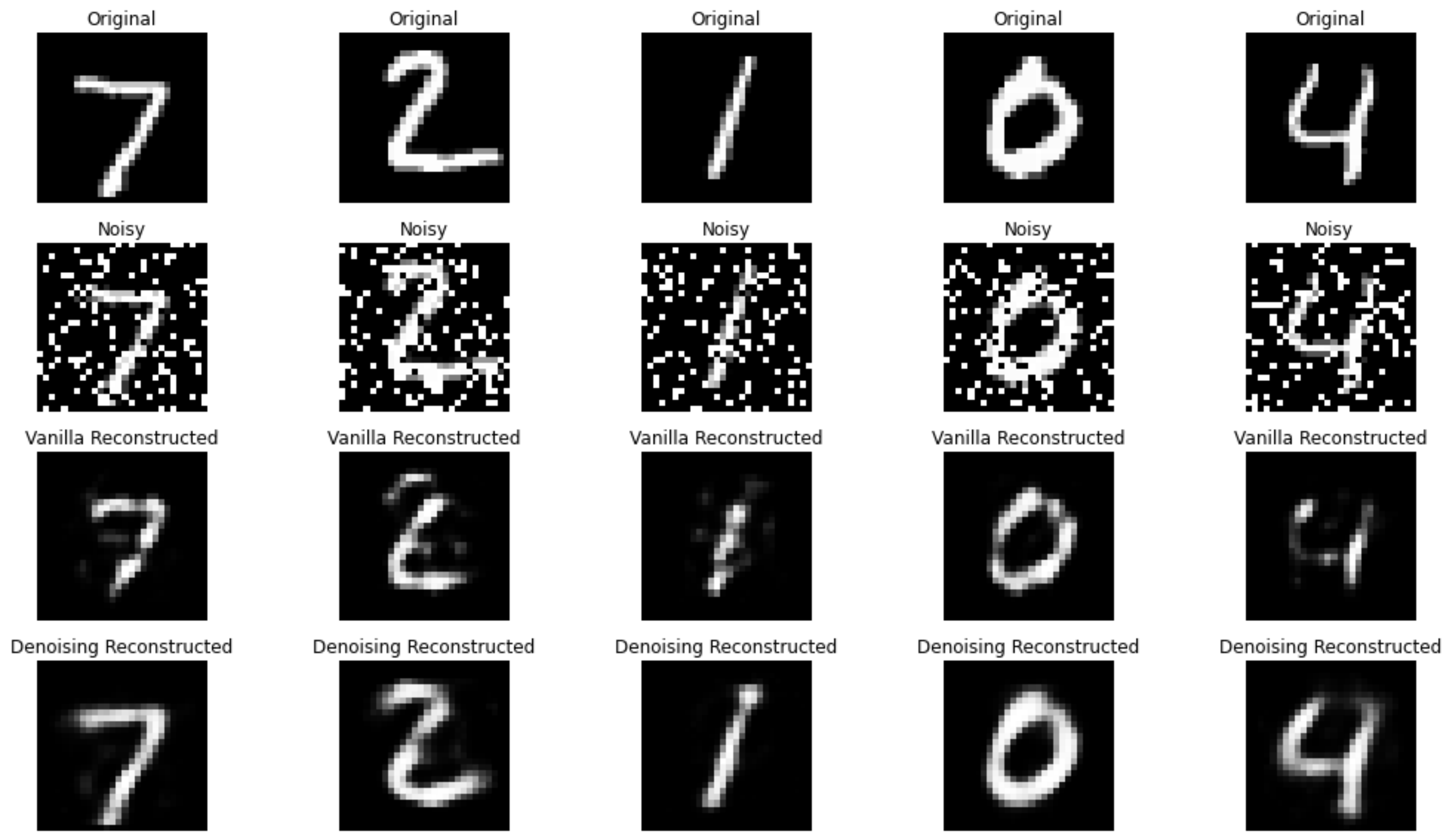
1. Minimize the reconstruction error between the clean input ($x$) and the reconstructed output ($\hat{x}$), despite noisy input ($x_{noisy}$).

$$Loss = \left|\left|x - \hat{x}\right|\right|_2^2$$

**_3. Advantages_**:

1. **Prevents trivial identity mapping**: By reconstructing from noisy input, the network avoids learning simple mappings.
2. **Robustness**: Improves the model's ability to generalize to unseen data or noisy scenarios.
3. **Dimensionality Reduction**: Produces meaningful lower-dimensional representations, useful for downstream tasks.

# Denoising Autoencoder



# Sparse Autoencoder

- A **Sparse Autoencoder** is a variant of the standard autoencoder that includes a **sparsity constraint** on the activations of the hidden layer (latent space). This constraint forces the model to activate only a small subset of neurons for a given input, encouraging the encoder to learn a more meaningful and efficient representation of the data.
- The sparsity constraint ensures that only a small fraction of neurons in the latent space are active (non-zero) for any given input. Let $h_j(x)$ represent the activation of the j-th neuron in the latent layer for input $x$. Define:

$$\boxed{\hat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N} h_j(x_i)}$$

where $\hat{\rho}_j$ is the average activation of the j-th neuron across the dataset.

- To enforce sparsity, we impose that $\hat{\rho}_j$ is close to a small target sparsity ρ (e.g., ρ=0.05, meaning 5% average activation). This is achieved using **KL Divergence**:

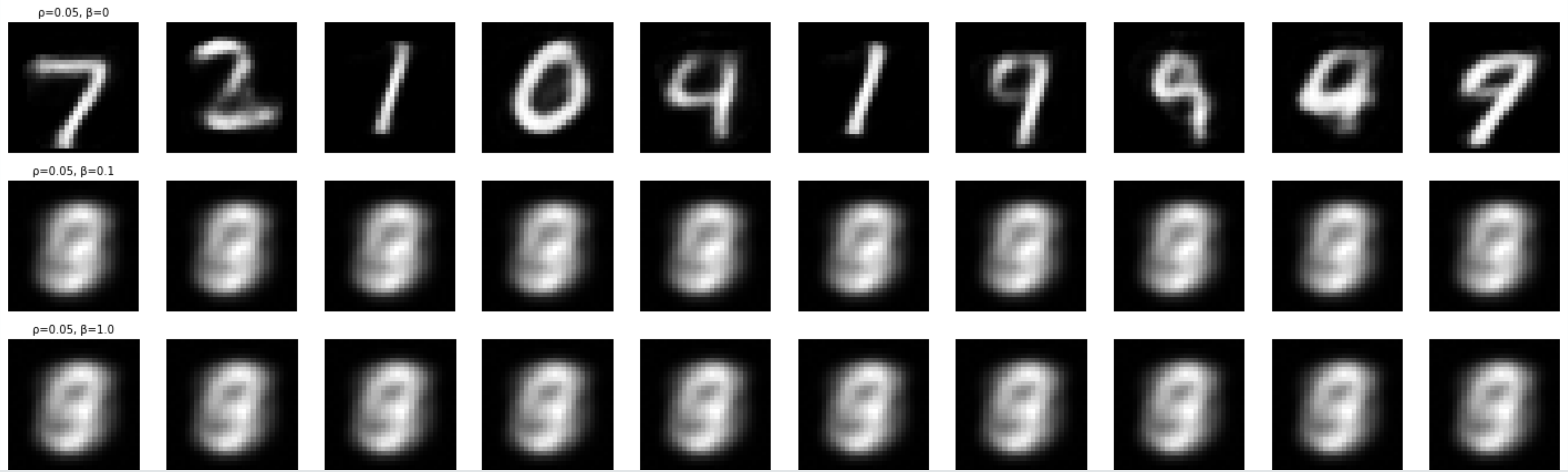
$$KL(\rho||\hat{\rho}_j) = \rho \log\left(\frac{\rho}{\hat{\rho}_j}\right) + (1-\rho)\log\left(\frac{1-\rho}{1-\hat{\rho}_j}\right)$$

- The total sparsity penalty is:

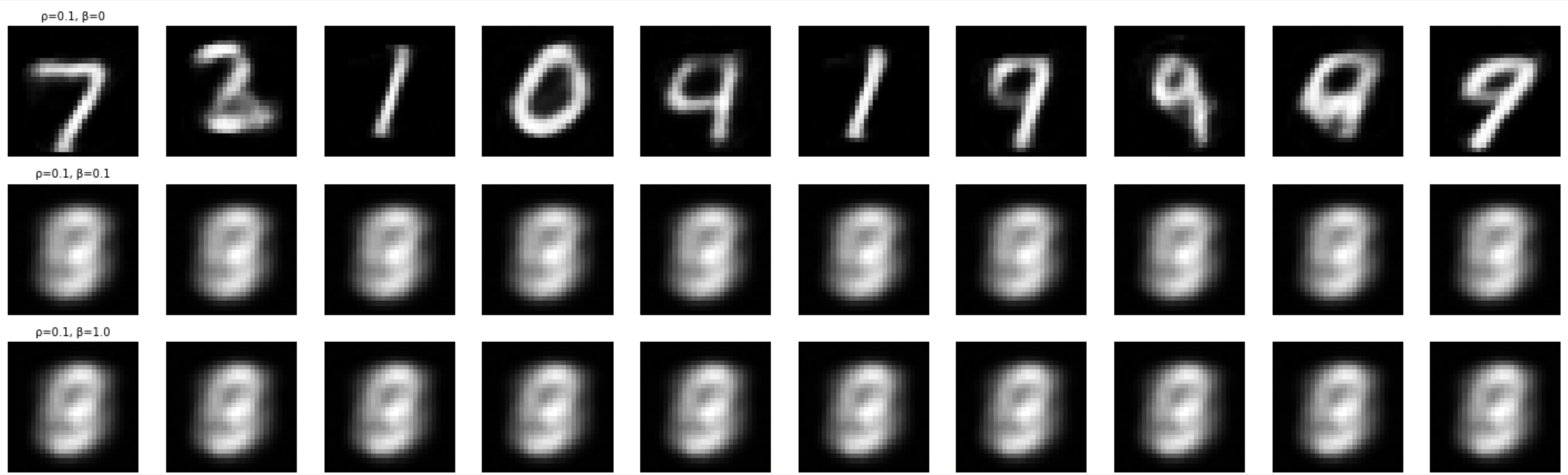
$$l_{sparse} = \beta \sum_{j=1}^{m} KL(\rho||\,\hat{\rho}_j) \quad + \|x - \hat{x}\|_2^2$$

Where $\beta$ controls the strength of the sparsity penalty, and m is the number of neurons in the latent layer.

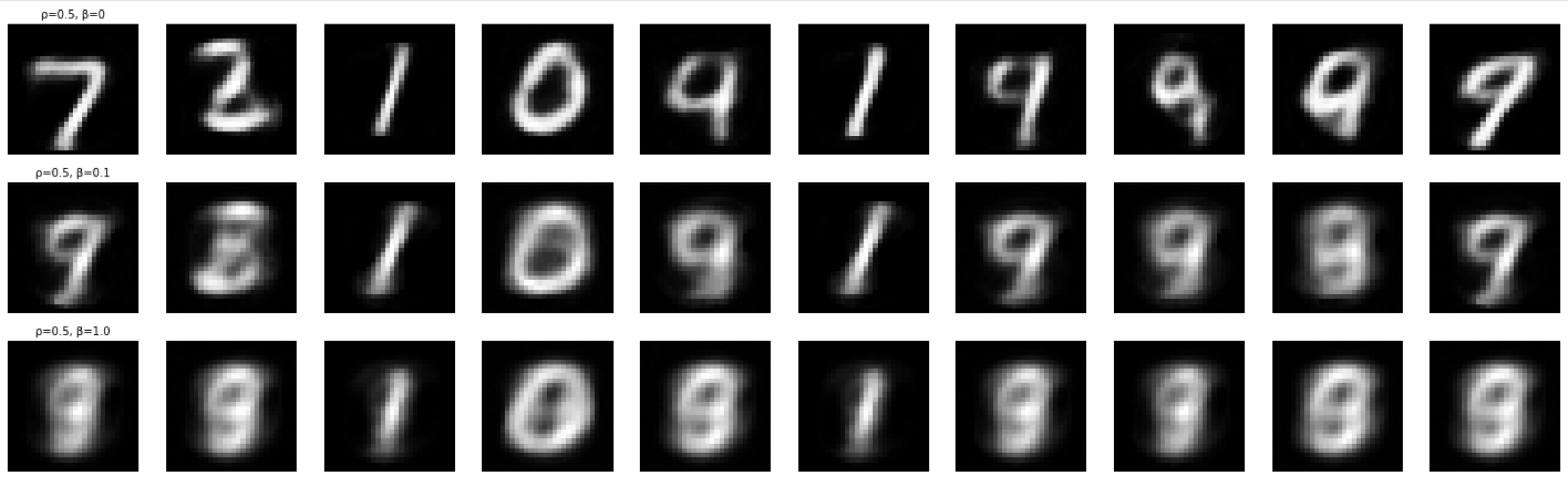
# *Sparse Autoencoder (0.05)*



# *Sparse Autoencoder (0.1)*



# Sparse Autoencoder (0.5)



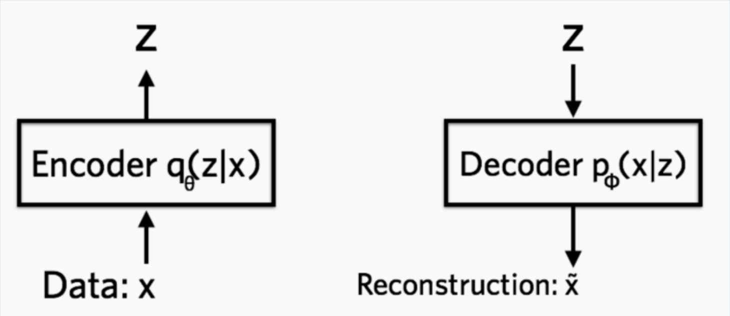
# *Variational Autoencoder*

- A **Variational Autoencoder (VAE)** is a type of **generative model**. Unlike traditional autoencoders that aim to reconstruct inputs deterministically, VAEs learn to encode input data into a **latent probability distribution**. This probabilistic nature enables VAEs to **generate new data samples** by sampling from the learned latent distribution.
- Key concepts in VAEs:

1. *The **encoder** maps input data to the parameters of a probability distribution (commonly Gaussian).*
2. *The **latent space** is modeled as a continuous probability distribution.*
3. *The **decoder** generates data samples by decoding points sampled from the latent distribution.*

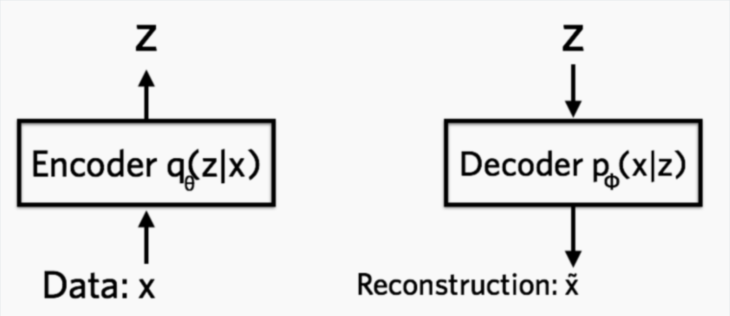
# *Variational Autoencoder*

- The encoder takes input and returns parameters for a probability density (e.g., Gaussian): I.e., $q_\theta(z \mid x)$ gives the mean and covariance matrix.
- We can sample from this distribution to get random values of the lower-dimensional representation *z*.
- Implemented via a neural network: each input *x* gives a vector mean and diagonal covariance matrix $q_\theta(z \mid x)$ that determine the Gaussian density
- Parameters $\theta$ for the NN need to be learned – need to set up a loss function.



# Variational Autoencoder

- The decoder takes latent variable *z* and returns parameters for a distribution. E.g., $p_\phi(x|z)$ gives the mean and variance for each pixel in the output.
- Reconstruction $\tilde{x}$ is produced by sampling.
- Implemented via neural network, the NN parameters $\phi$ are learned.

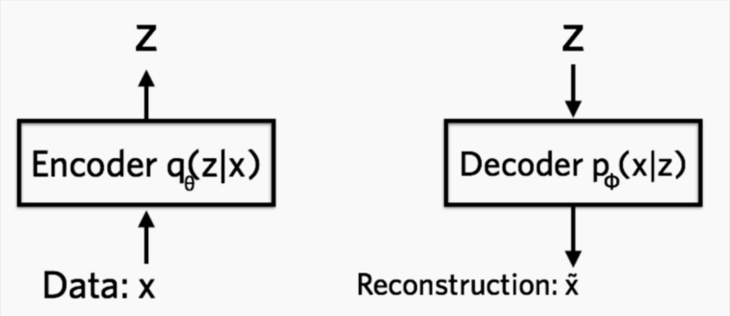


# *Variational Autoencoder*

- Loss function for autoencoder: $L_2$ distance between output and input (or clean input for denoising case)
- For VAE, we need to learn parameters of two probability distributions. For a single input, $x_i$, we maximize the expected value of returning $x_i$ or minimize the expected negative log likelihood.

$$-\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)]$$

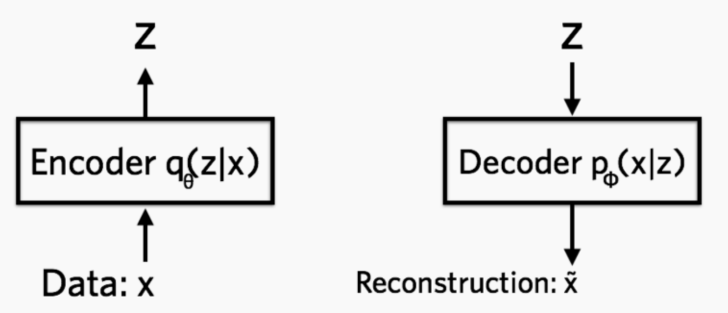
- This takes expected value wrt $z$ over the current distribution $q_\theta(z|x_i)$ of the loss $-\log p_\phi(x_i|z)$



# Variational Autoencoder

- For a single data point $x_i$ we get the loss function

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$




- The first term promotes recovery of the input.
- The second term keeps the encoding continuous – the encoding is compared to a fixed $p(z)$ regardless of the input, which inhibits memorization.
- With this loss function the VAE can (almost) be trained using gradient descent on minibatches.

# Variational Autoencoder

- After training, $q_\theta(z|x_i)$ is close to a standard normal, N(0,1) – easy to sample.
- Using a sample of *z* from $q_\theta(z|x_i)$ as input to sample from $p_\phi(x|z)$ gives an approximate reconstruction of $x_i$, at least in expectation.
- If we sample any *z* from N(0,1) and use it as input to to sample from $p_\phi(x|z)$ then we can approximate the entire data distribution $p(x)$. I.e., we can generate new samples that look like the input but aren't in the input.