

CSE/ISyE 6740
Computational Data Analysis

Autoencoder and Variational Autoencoder

10/21/2026

Kai Wang, Assistant Professor in Computational Science and Engineering
kwang692@gatech.edu

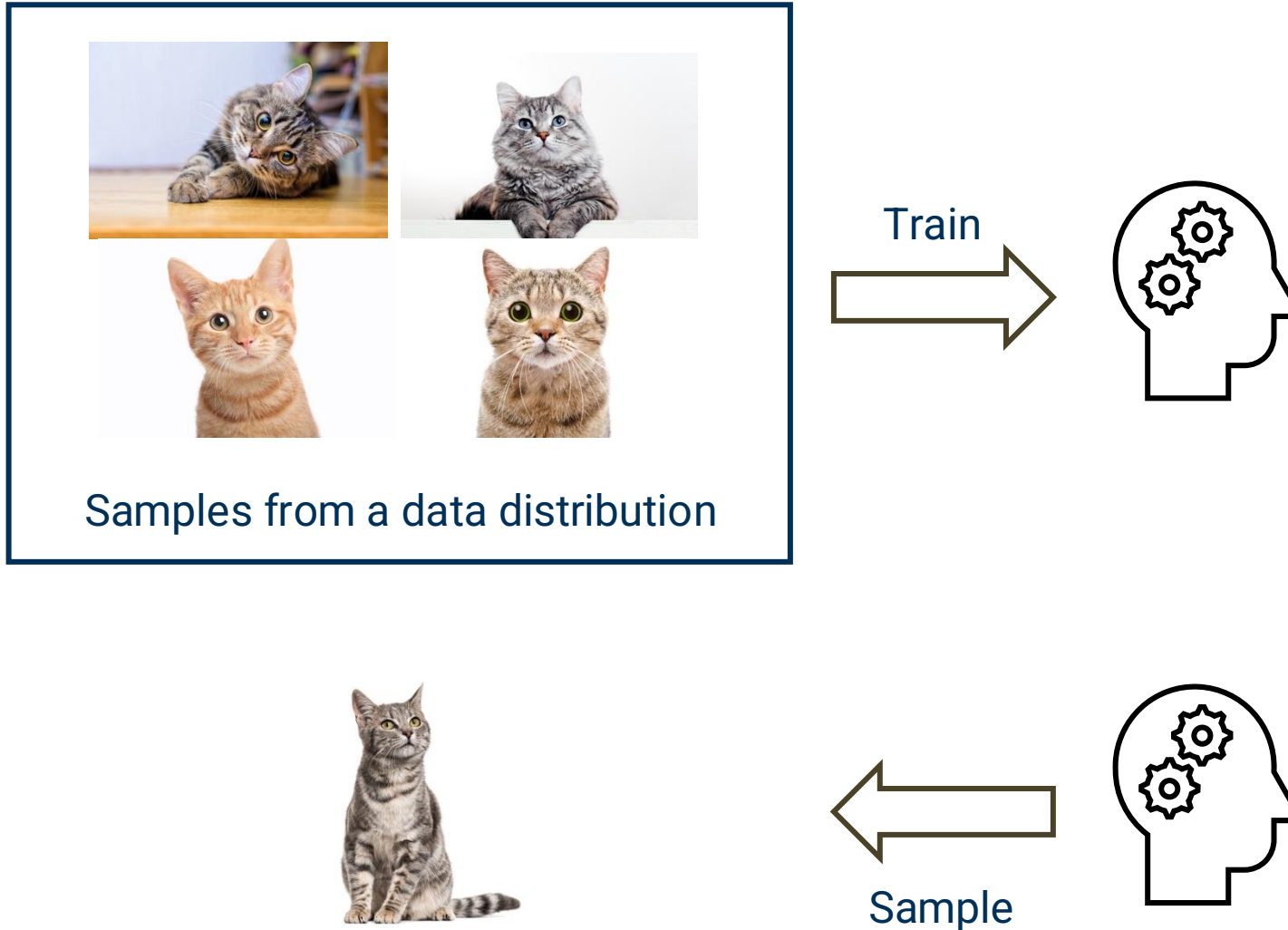
Outline

- **Advanced Machine Learning**

- Neural networks
 - Generative models
 - Autoencoders
 - Variational autoencoders
 - Image generation

Generative Models

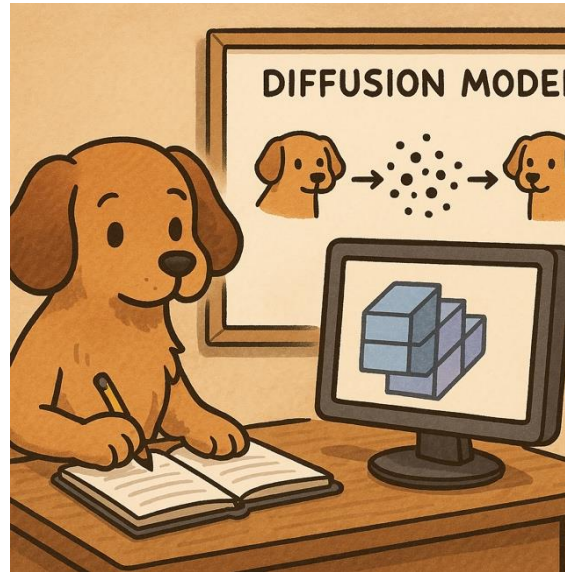
What Do We Want From Generative Models?



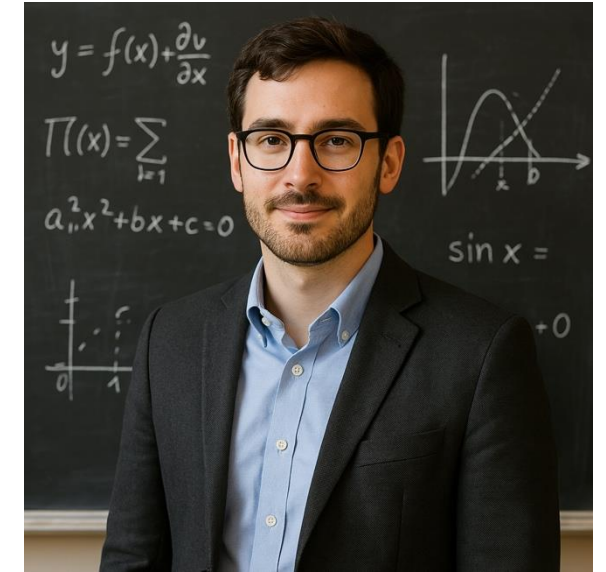
Applications



Convert the TA's image into Ghibli style



A dog learning diffusion model



Generate a selfie of me based on your impression of our conversations so far

Generative AI Companies



Leonardo.Ai



Imagen

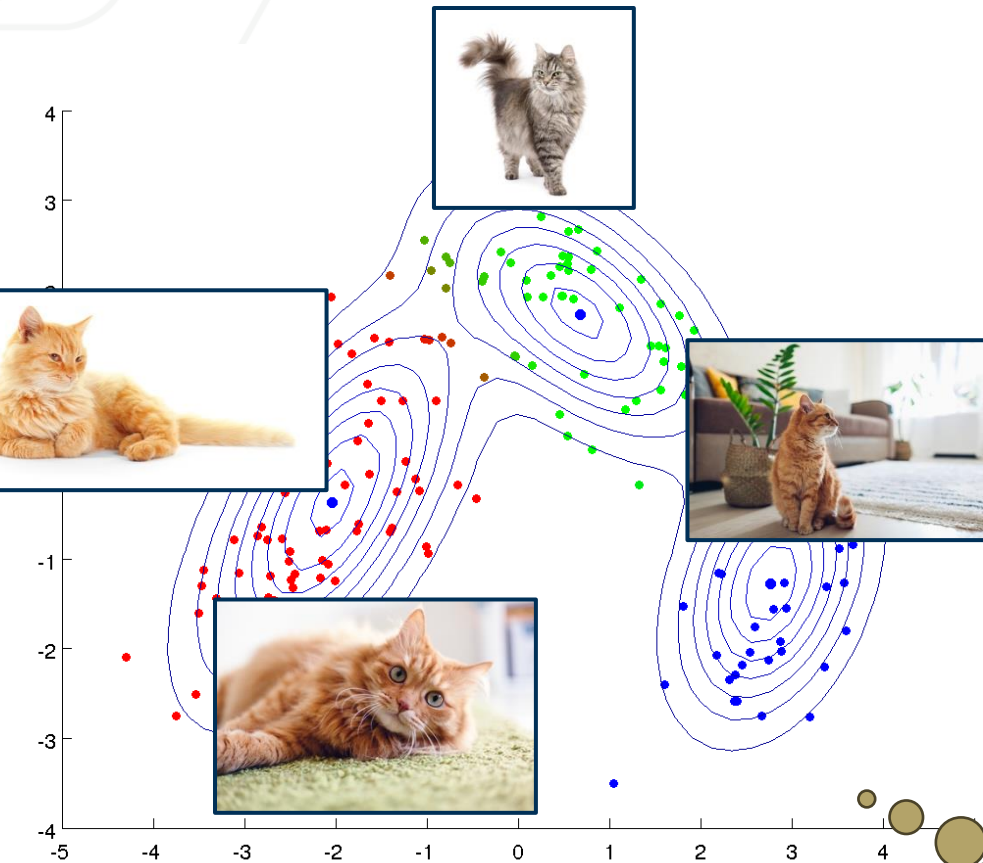
Our highest quality text-to-image model

[Try in Gemini >](#)

[Try in ImageFX >](#)



Generative Models: NNs for Density Estimation



Density estimation



Find the parameter θ to maximize the likelihood $\log P(D|\theta)$

$$p(x | \theta) = \sum_i \pi_i N(x|\mu_i, \Sigma_i)$$



Generation

Given the parameter θ , generate a sample x

$$x \sim p(\cdot | \theta) = \sum_i \pi_i N(\cdot | \mu_i, \Sigma_i)$$

How to extend to high-dimensional data?

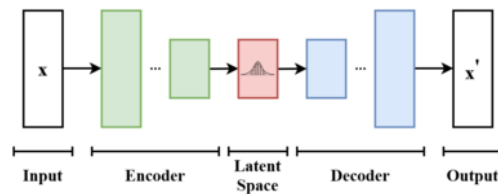
History of Generative Models

Before 2012

No effective generative models

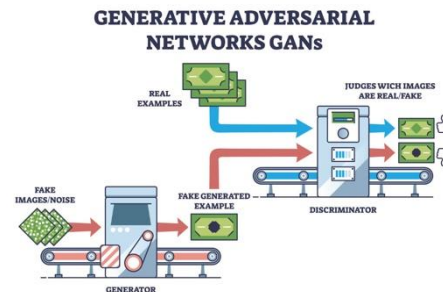
2013

Variational autoencoder



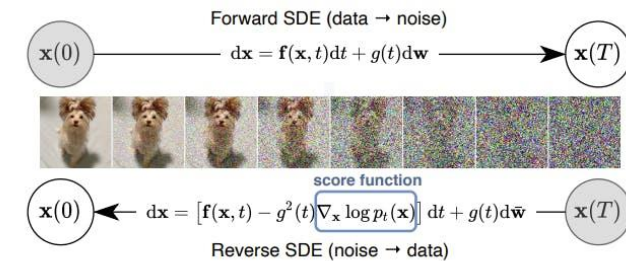
2014

Generative Adversarial Network



2020

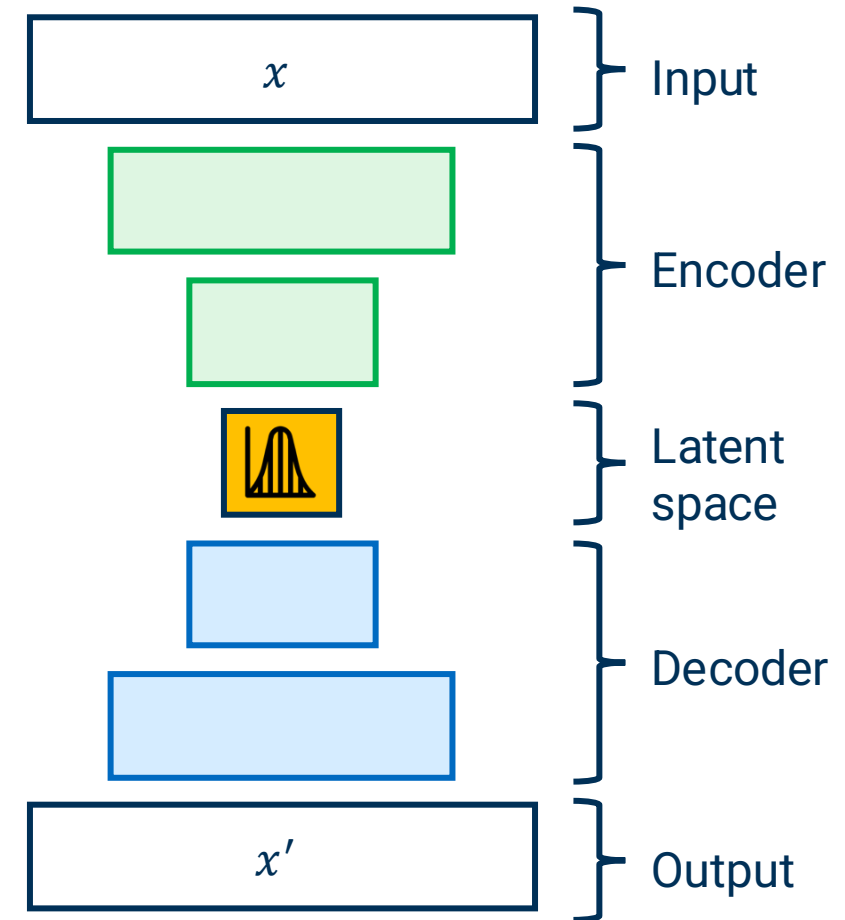
Diffusion models



Autoencoder

Autoencoder

- An **autoencoder** is composed of a decoder and an encoder, each parameterized by a deep neural network.
- The goal is to take an input x to generate a latent embedding z , and later predict x back.
- To make this goal non-trivial, we need to add a **bottleneck layer** whose dimension is much smaller than the input.

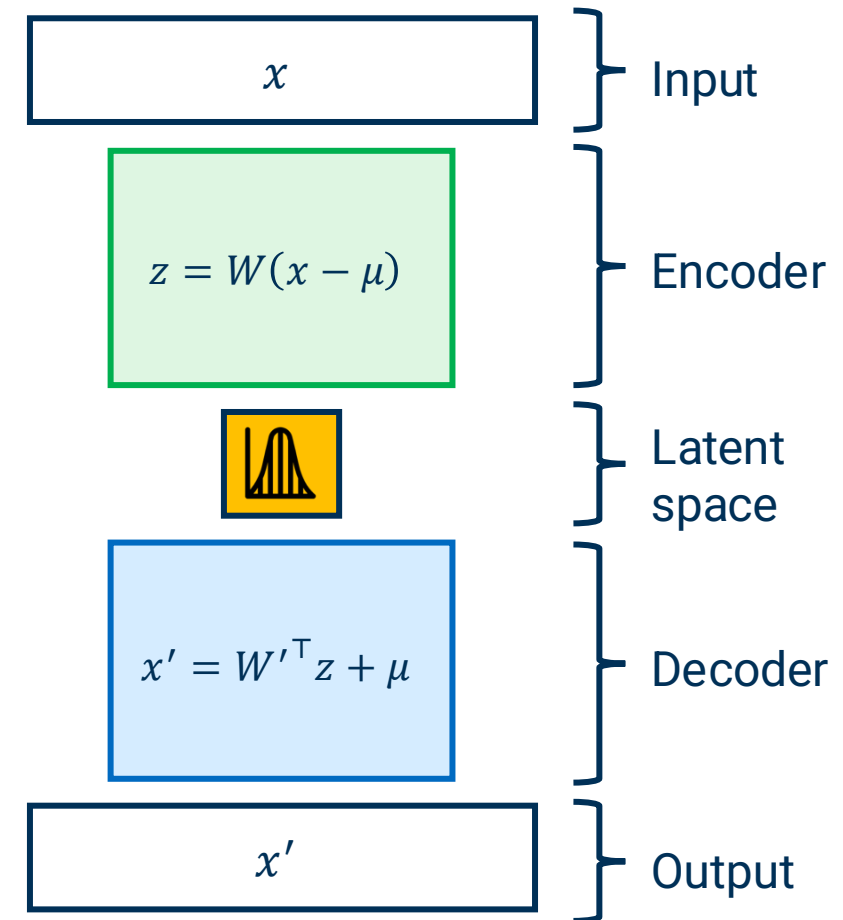


PCA as an Autoencoder

- PCA is a special case of autoencoder with one-layer NN and no activation function.

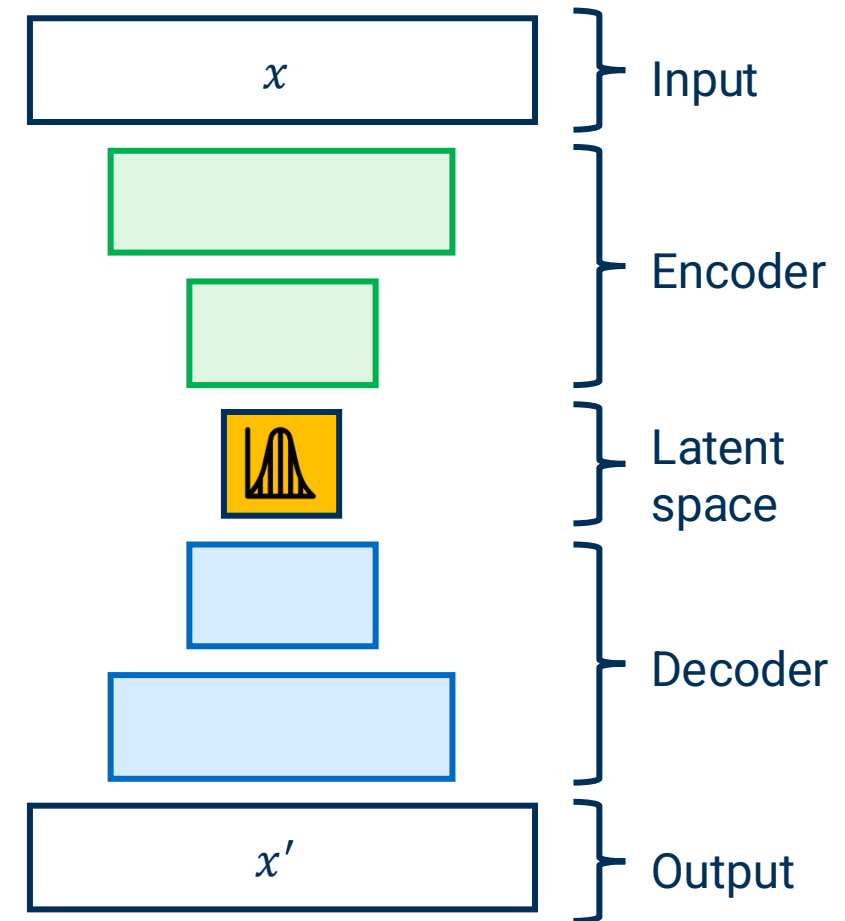
$$z^i = \begin{pmatrix} w^{1\top} (x^i - \mu) / \sqrt{\lambda_1} \\ w^{2\top} (x^i - \mu) / \sqrt{\lambda_2} \\ \vdots \\ w^{d'\top} (x^i - \mu) / \sqrt{\lambda_{d'}} \end{pmatrix}$$

$$x^{i'} = \mu + z_1^i \cdot \sqrt{\lambda_1} \cdot w^1 + z_2^i \cdot \sqrt{\lambda_2} \cdot w^2 + \dots$$



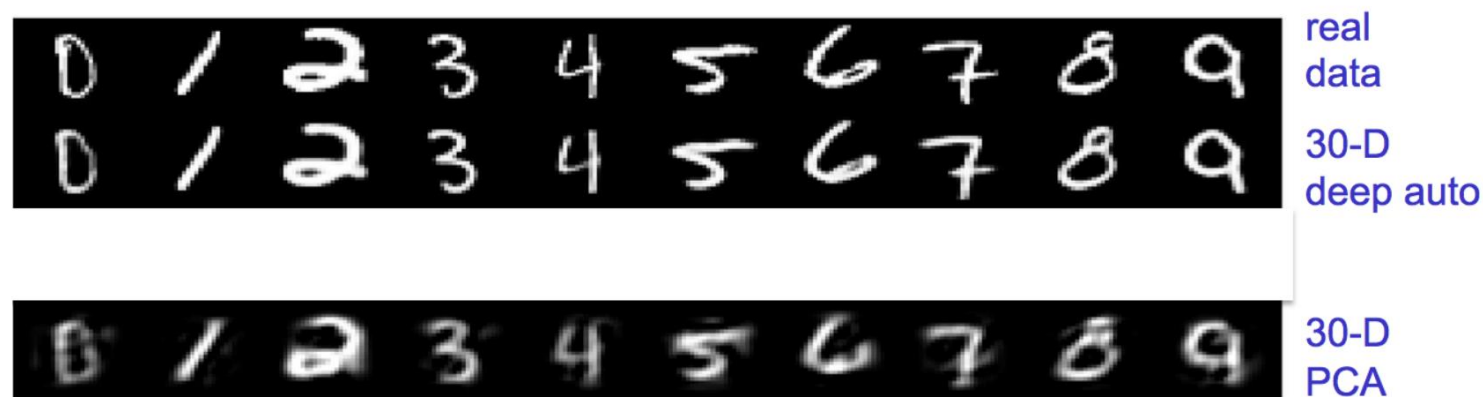
Architecture for Autoencoders

- What architecture should we use for the **encoder**?
- What architecture should we use for the **decoder**?



Deep Autoencoder

- Shallow one-layer autoencoder = principal component analysis
- Deep autoencoders = non-linear dimensionality reduction

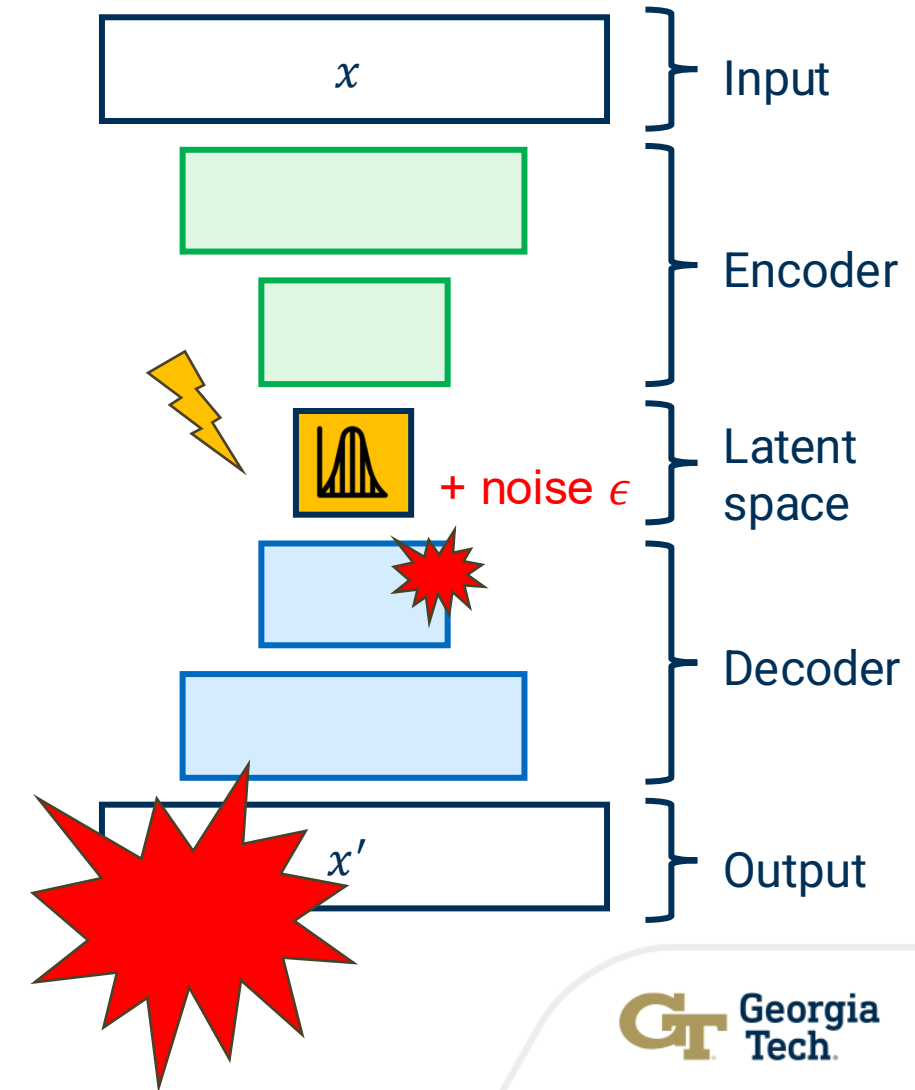


Some limitations of autoencoders

- They're not generative models, so they don't define a distribution
- How to choose the latent dimension?

Deep Autoencoder

- Injecting noise to the latent space can corrupt the whole decoding process.
- Think about modifying your zip file and see if you can unzip it.
- The injected noise can get amplified through the decoding process and lead to meaningless output.



Variational Autoencoder

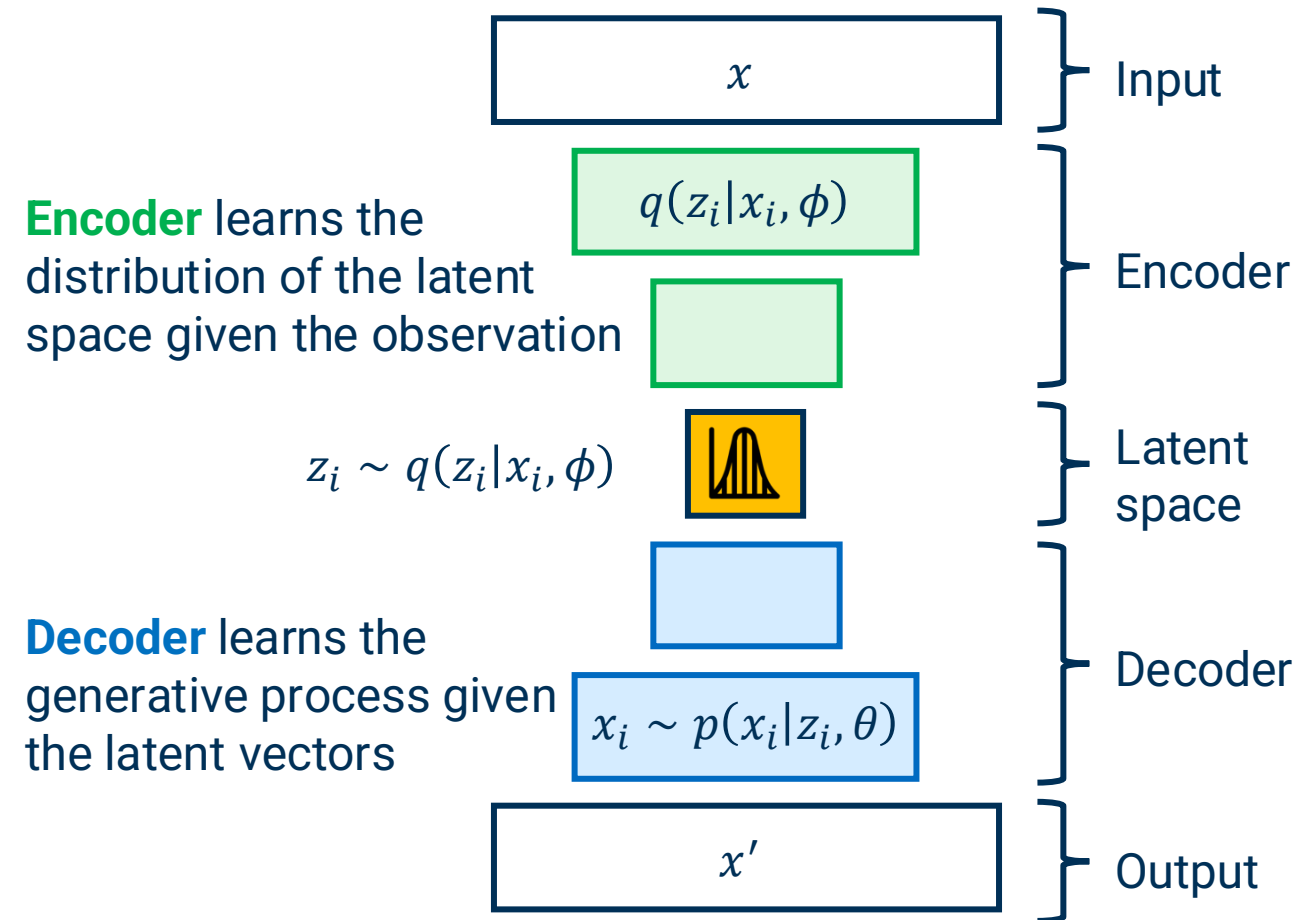
Variational Autoencoder

- **Encoder:** a probabilistic model by predicting the mean and the variance.
- **Decoder:** a probabilistic model by predicting the mean

- The latent variable:
$$z_i \sim q(z_i|x_i, \phi)$$
$$= \mathcal{N}(\mu_\phi(x_i), \sigma_\phi^2(x_i))$$

Loss function:

$$L = \sum_i \frac{1}{2\eta} \|x_i - G_\theta(z_i)\|^2 + D_{KL}(\underbrace{q(z)}_{\mathcal{N}(\mu_\phi(x_i), \sigma_\phi^2(x_i))} || \underbrace{p(z)}_{\mathcal{N}(0,1)})$$



Variational Autoencoder Loss

- Consider training a generator network with maximum likelihood

$$p(x) = \int p(z) p(x|z) dz$$

- **One problem:** if z is low-dimensional and the decoder is deterministic, then $p(x) = 0$ almost everywhere!
- **Solution:** define a noisy observation, e.g.,

$$p(x|z) = \mathcal{N}(x; G_{\theta}(z), \eta I)$$

Variational Autoencoder Loss

$$\begin{aligned}\log p(x) &= \log \int p(z) p(x|z) dz \\ &= \log \int q(z) \frac{p(z)}{q(z)} p(x|z) dz \\ &\geq \int q(z) \log \left(\frac{p(z)}{q(z)} p(x|z) \right) dz \quad (\text{Jensen inequality}) \\ &= \mathbb{E}_q \left[\log \frac{p(z)}{q(z)} \right] + \mathbb{E}_q [\log p(x|z)]\end{aligned}$$

KL divergence loss between q and p

- We pick $p = \mathcal{N}(0,1)$, $q = \mathcal{N}(\mu_\phi(x_i), \sigma_\phi^2(x_i))$

```
def loss_function(self,
                  *args,
                  **kwargs) -> dict:
    self.num_iter += 1
    recons = args[0]
    input = args[1]
    mu = args[2]
    log_var = args[3]
    kld_weight = kwargs['M_N'] # Account for the minibatch samples from the dataset

    recons_loss = F.mse_loss(recons, input)

    kld_loss = torch.mean(-0.5 * torch.sum(1 + log_var - mu ** 2 - log_var.exp(), dim = 1), dim = 0)
```

Reconstruction loss

$$\begin{aligned}p(x|z) &= \mathcal{N}(x; G_\theta(z), \eta I) \\ \mathbb{E}_q [\log p(x|z)] &= -\frac{1}{2\eta} \|x - G_\theta(z)\|^2\end{aligned}$$

Intuitions Push the latent distribution $\mathcal{N}(\mu_\phi(x_i), \sigma_\phi^2(x_i))$ generated from the encoder closer to a normal distribution

Push the reconstruction $G_\theta(z)$ closer to the original input x

Questions

- Why $\mathbb{E}_q[\log p(x|z)] = -\frac{1}{2\eta} \|x - G_\theta(z)\|^2$

- How to compute $\mathbb{E}_q \left[\log \frac{p(z)}{q(z)} \right]$
 - $p = \mathcal{N}(0,1)$
 - $q = \mathcal{N}(\mu_\phi(x_i), \sigma_\phi^2(x_i))$

```
def loss_function(self,
                    *args,
                    **kwargs) -> dict:
    self.num_iter += 1
    recons = args[0]
    input = args[1]
    mu = args[2]
    log_var = args[3]
    kld_weight = kwargs['M_N'] # Account for the minibatch samples from the dataset

    recons_loss = F.mse_loss(recons, input)

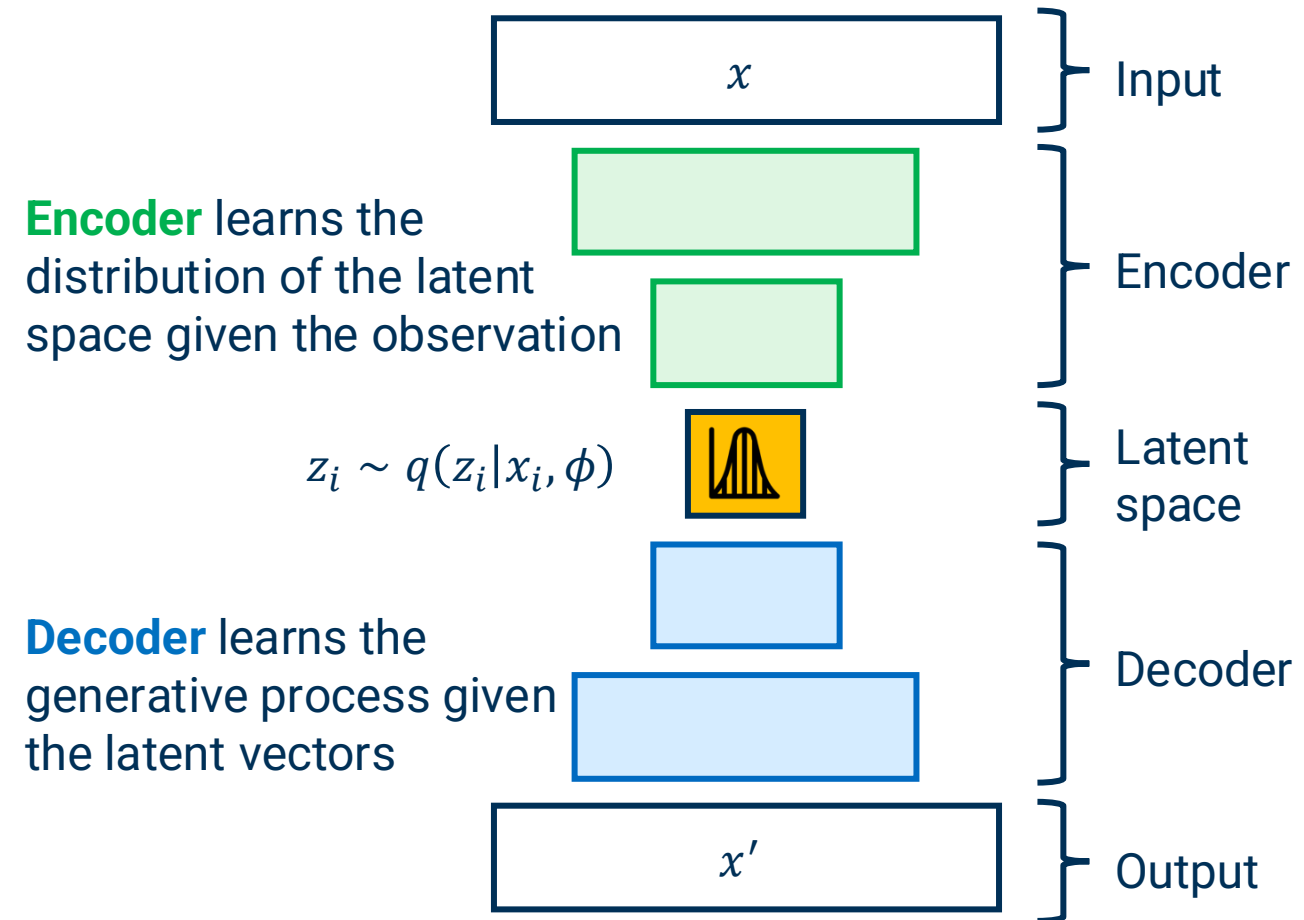
    kld_loss = torch.mean(-0.5 * torch.sum(1 + log_var - mu ** 2 - log_var.exp(), dim = 1), dim = 0)
```

Variational Autoencoder Loss

- Loss function has two terms

$$L = \sum_i \frac{1}{2\eta} \|x_i - G_\theta(z_i)\|^2 + D_{KL}(q(z)||p(z))$$

- KL divergence loss:** measure the difference between the generated latent distribution $q(z)$ and the target latent distribution $p(z)$
- Reconstruction loss:** measure how the reconstructed $x' = G_\theta(z)$ differs from the original x



Variational Autoencoder

- PyTorch implementation
 - **Encoder**
 - One NN to predict mean
 - One NN to predict log variance (to compute the KL divergence of two Gaussian distributions)
 - **Decoder**
 - One NN to represent the mean
 - Variance is assumed to be 1

```
def encode(self, input: Tensor) -> List[Tensor]:
    """
    Encodes the input by passing through the encoder network
    and returns the latent codes.
    :param input: (Tensor) Input tensor to encoder [N x C x H x W]
    :return: (Tensor) List of latent codes
    """
    result = self.encoder(input)
    result = torch.flatten(result, start_dim=1)

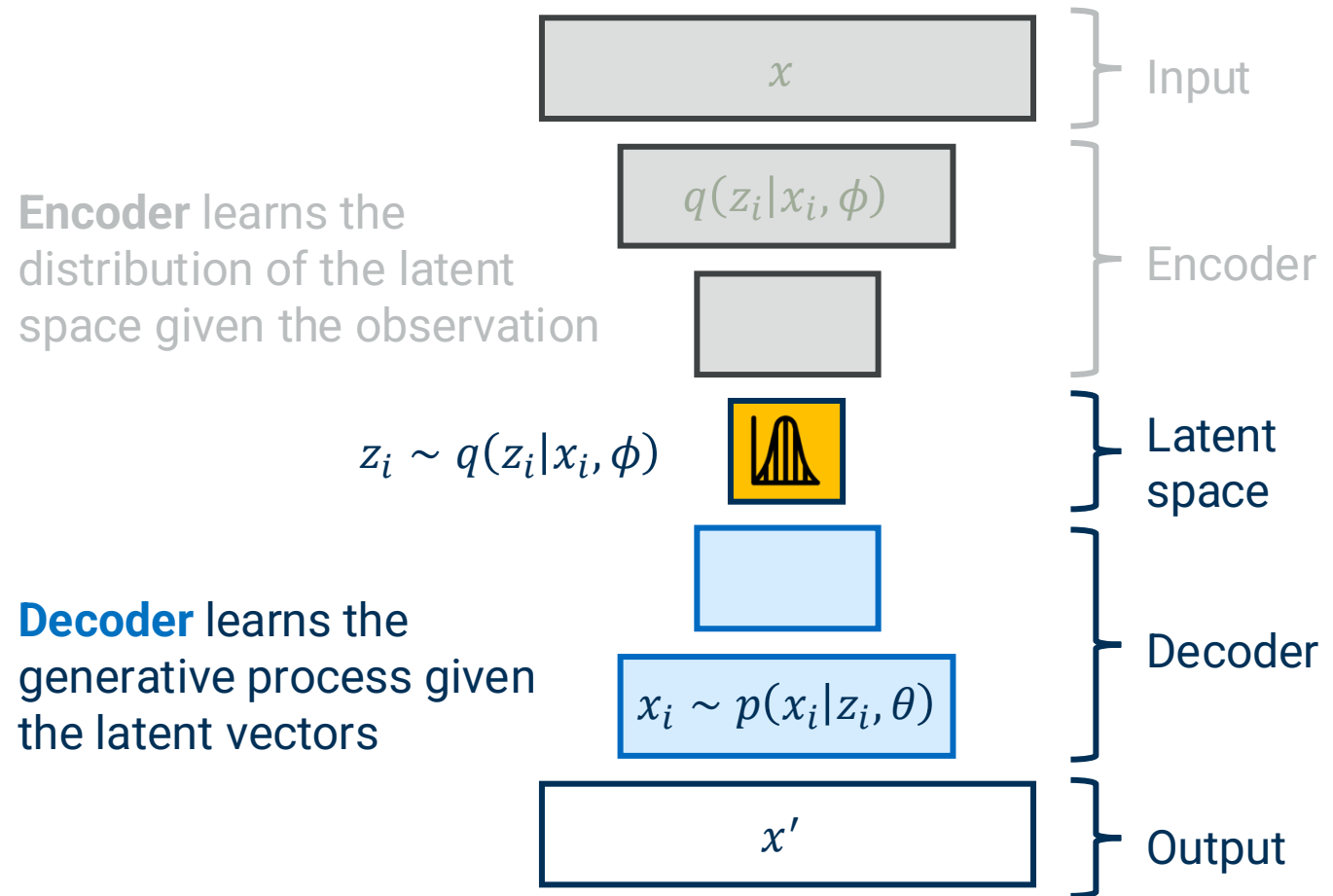
    # Split the result into mu and var components
    # of the latent Gaussian distribution
    mu = self.fc_mu(result)
    log_var = self.fc_var(result)

    return [mu, log_var]

def decode(self, z: Tensor) -> Tensor:
    """
    Maps the given latent codes
    onto the image space.
    :param z: (Tensor) [B x D]
    :return: (Tensor) [B x C x H x W]
    """
    result = self.decoder_input(z)
    result = result.view(-1, 512, 2, 2)
    result = self.decoder(result)
    result = self.final_layer(result)
    return result
```

Applications of VAE

- The decoder part of the VAE is particularly useful for generative AI
- Randomly generate a latent variable $z \sim \mathcal{N}(0,1)$
- Feed into the decoder to reconstruct a realistic input $x' \sim p(x|z, \theta)$



Applications of VAE

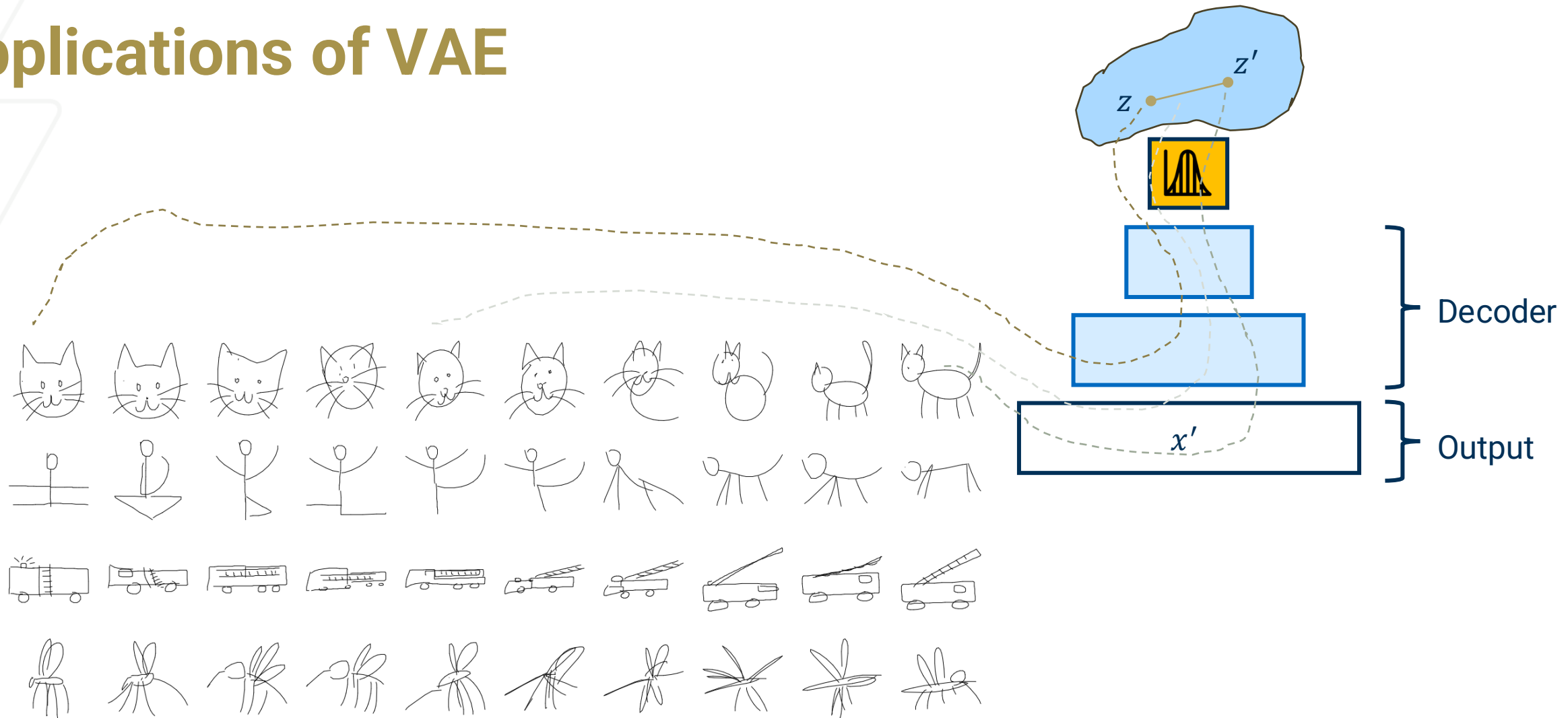


Figure 1: Latent space interpolation of various vector images produced by our model.

Conditional VAE

- If we have additional information about the class label (e.g., a cat image), then we can add the context to both encoder and decoder.
- We train the conditional encoder and decoder as vanilla VAE.

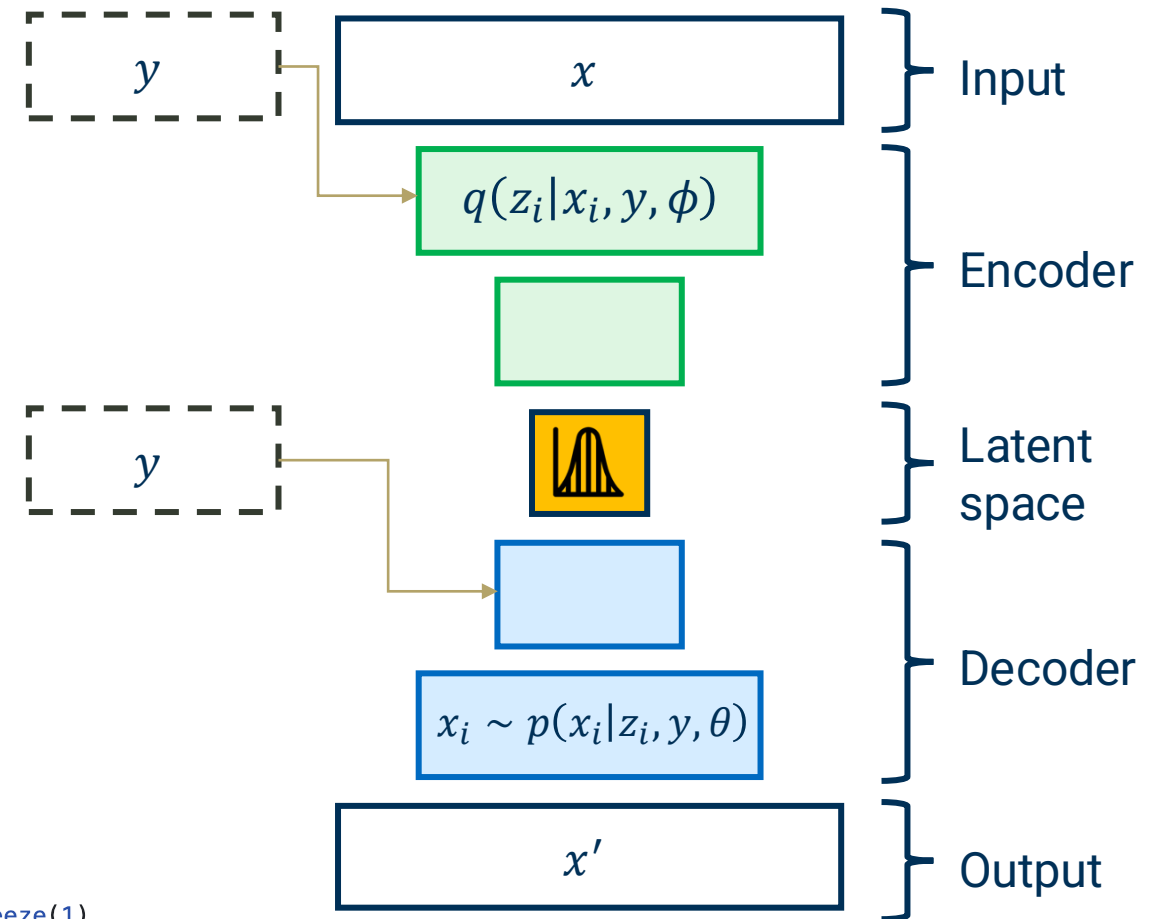
```
def forward(self, input: Tensor, **kwargs) -> List[Tensor]:  
    y = kwargs['labels'].float()  
    embedded_class = self.embed_class(y)  
    embedded_class = embedded_class.view(-1, self.img_size, self.img_size).unsqueeze(1)  
    embedded_input = self.embed_data(input)
```

```
x = torch.cat([embedded_input, embedded_class], dim = 1)  
mu, log_var = self.encode(x)
```

```
z = self.reparameterize(mu, log_var)
```

```
z = torch.cat([z, y], dim = 1)  
return [self.decode(z), input, mu, log_var]
```

Context information



Applications: style transfer, causal inference

“Semi-Supervised Learning with Deep Generative Models”

Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, Max Welling, NeurIPS 2014

<https://github.com/AntixK/PyTorch-VAE/blob/master/models/cvae.py>