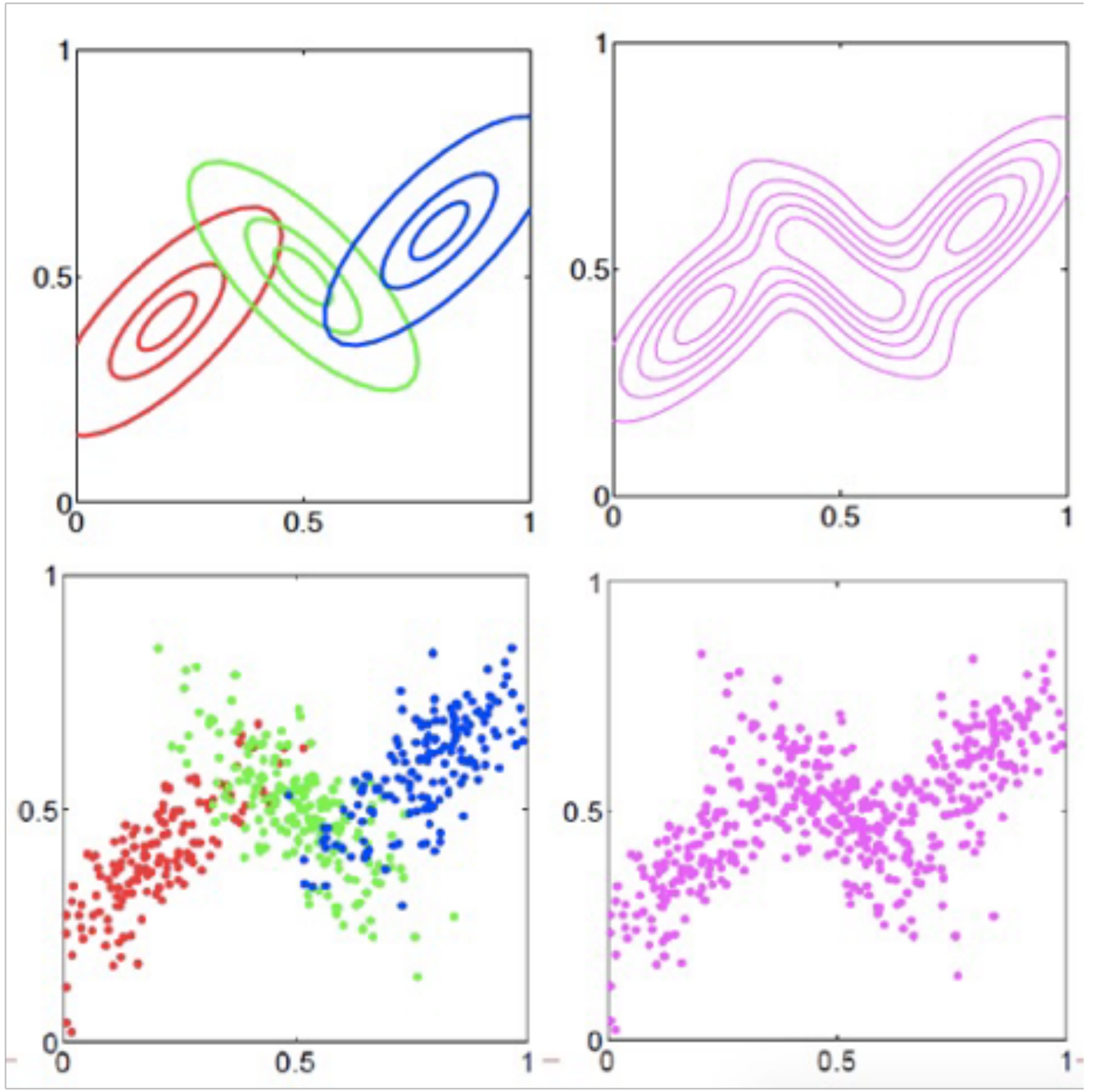
Bits of Vision: Generative Modeling - 1

Recap

- **Generative modeling.** Using unlabeled training data $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim p_{\text{data}}(\mathbf{x})$ to approximate the true data-generating distribution, such that:

$$p_\theta(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$$

- **Example.** Gaussian mixture models
  - Cluster information helps getting high accuracy in downstream tasks
  - Helps evaluating how likely each data point is — anomaly detection



Generative Modeling

- In modern contexts, generative modeling is being used as-is.
- **Example.** Suppose that we have a good model on the joint distribution:

$$p_\theta(\mathbf{x}, y) \approx p_{\text{data}}(\mathbf{x}, y)$$

from the image-text pairs $\{(\mathbf{x}, y)\}_{i=1}^{n}$ crawled from the web

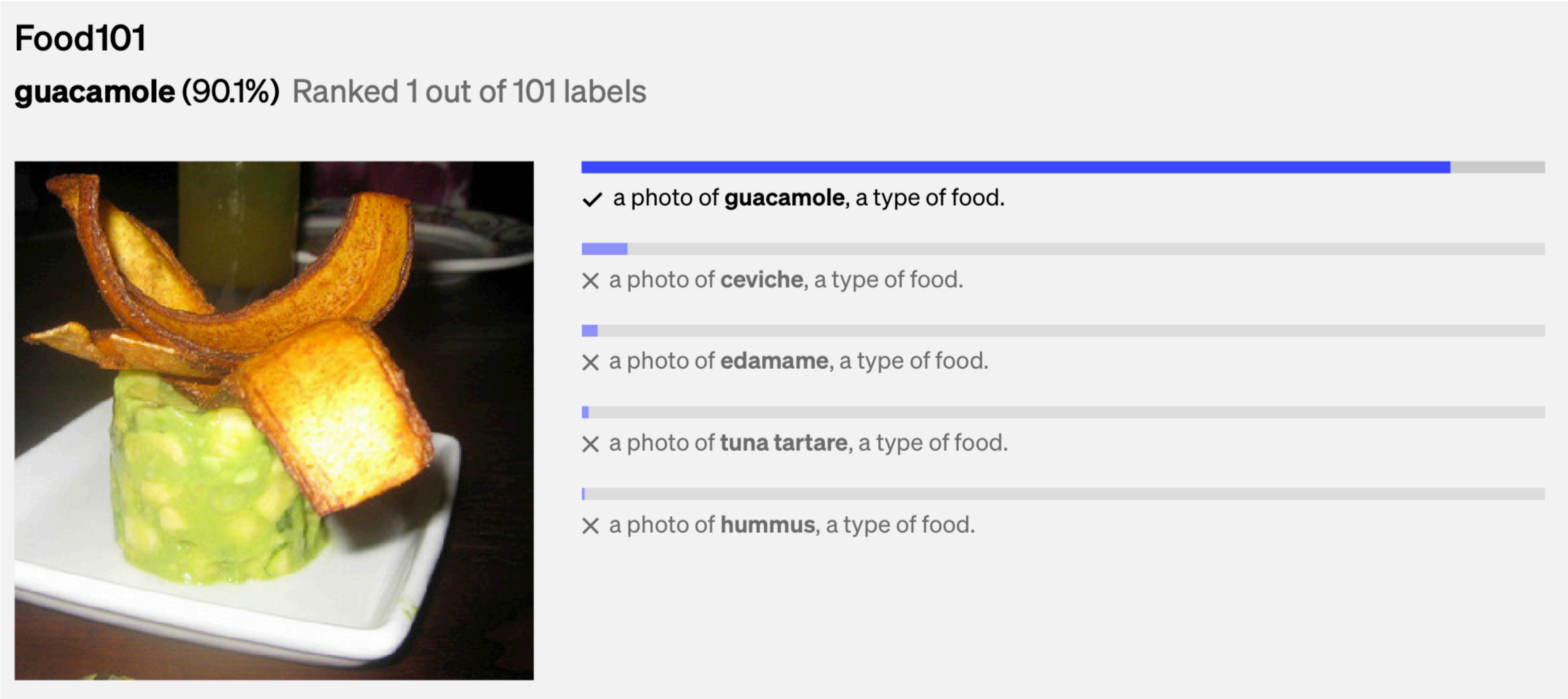
  - Treating $\mathbf{z} = (\mathbf{x}, y)$ as the unlabeled data

OpenAI “CLIP”

Generative Modeling

- With a good generative model, we can do the following things:
- **Generative Image Classification.** We can use the Bayes rule to do:

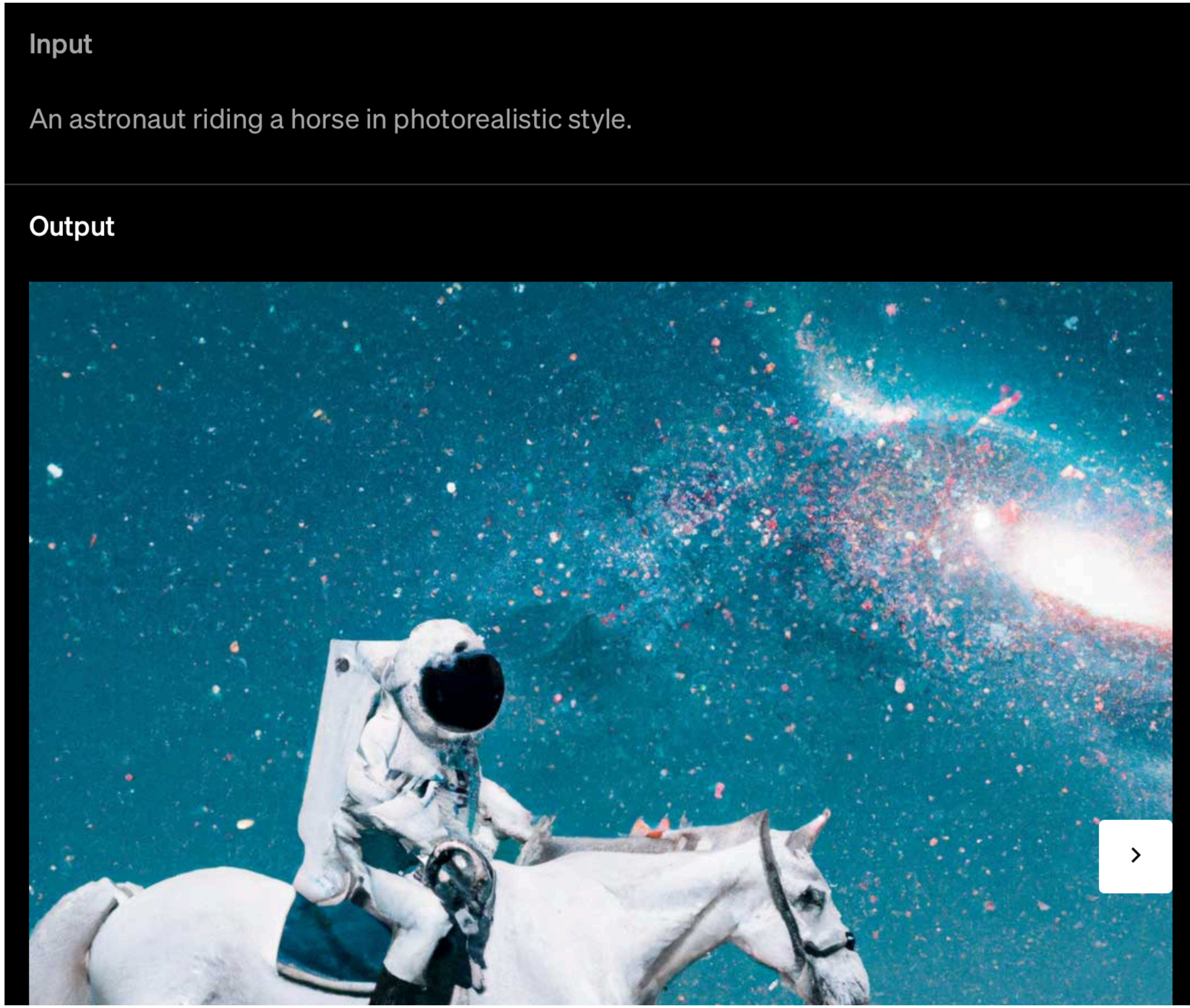
$$p_\theta(y \mid \mathbf{x}) = \frac{p_\theta(\mathbf{x}, y)}{p_\theta(\mathbf{x})}$$



Generative Modeling

- **Text-conditioned Generation.** Use the Bayes rule the other way, to generate an image from a text description

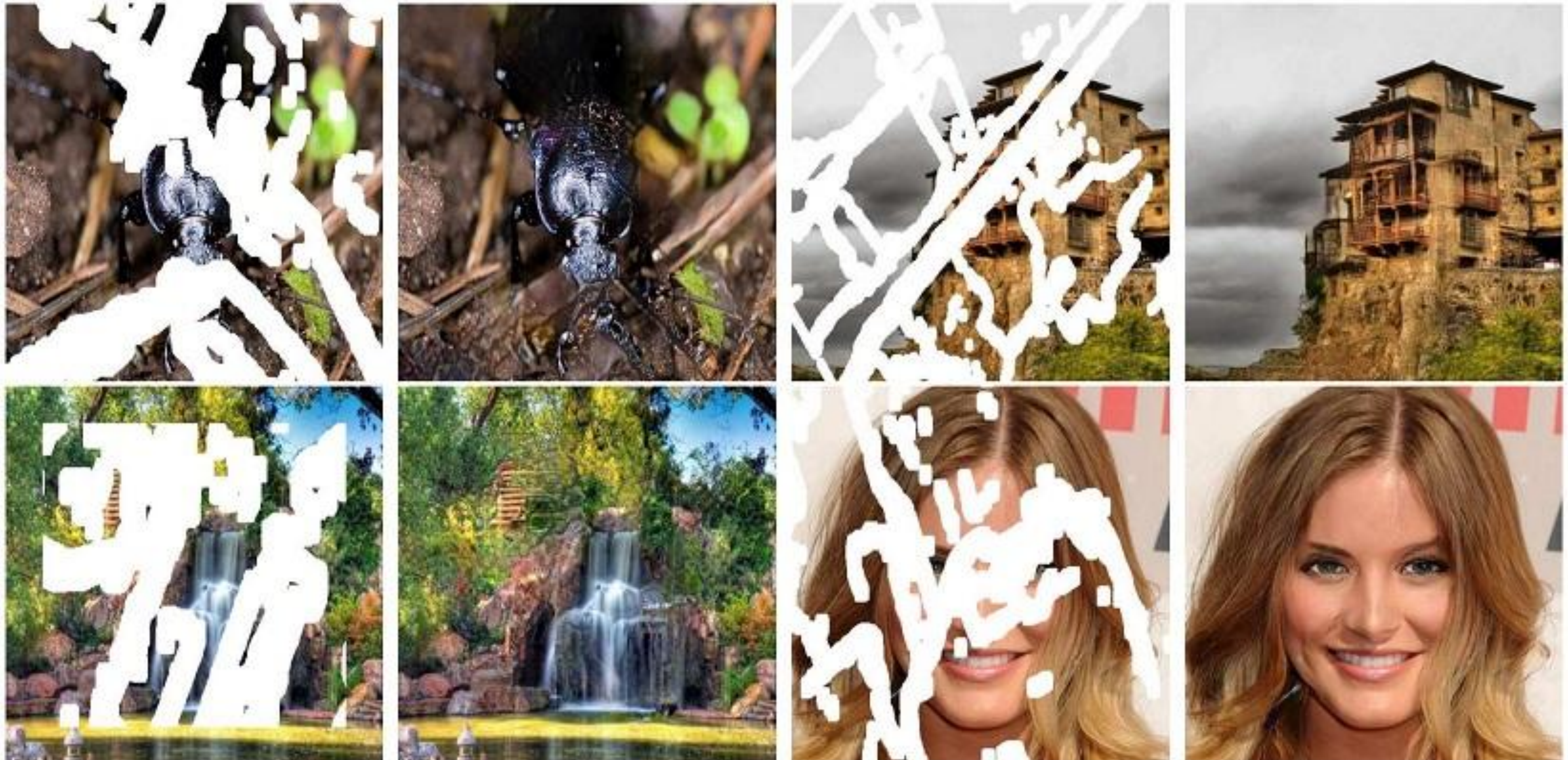
$$p_\theta(\mathbf{x} \mid y) = \frac{p_\theta(\mathbf{x}, y)}{p_\theta(y)}$$



Generative Modeling

- **Image Inpainting.** Generate missing pixels of the image in a way that they are well-aligned with the observed pixels

$$p_\theta(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d)$$



Generative Modeling

- **Text Generation.** Generate the next word that is most likely to follow the given text prompt

$$p_\theta(y_{n+1} \mid y_1, \ldots, y_n)$$

```rust
pub struct EncodedMessageQueue {
    pub queue: Vec<(EncodedMessage, ClientFilter)>,

    sender: Arc<Sender<Vec<(EncodedMessage, ClientFilter)>>>,
    receiver: Arc<Receiver<Vec<(EncodedMessage, ClientFilter)>>>,
}

impl EncodedMessageQueue {      You, 8 seconds ago • Uncommitted changes
    pub fn new() -> Self {
        let (sender, receiver) = crossbeam_channel::unbounded();
        Self {
            queue: vec![],
            sender: Arc::new(sender),
            receiver: Arc::new(receiver),
        }
    }
}
```

Today

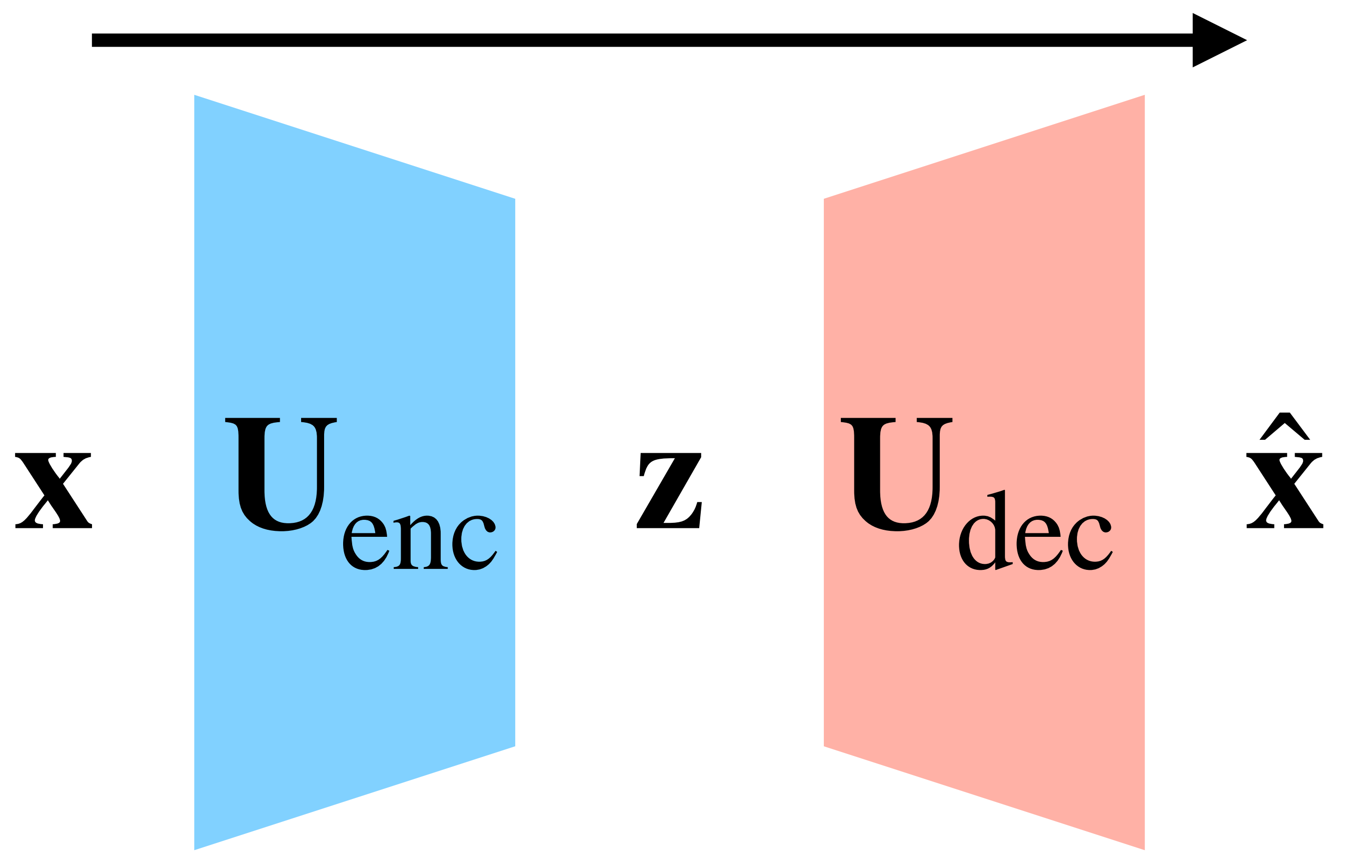
- Focus on the generative modeling for images

- **Comparison with Language.**
  - Need to generate many pixels for high-resolution images
    - Thus challenging to generate “realistic” ones
  - Pixels have mostly continuous values, not discrete
  - More locality involved, with 2D/3D geometry

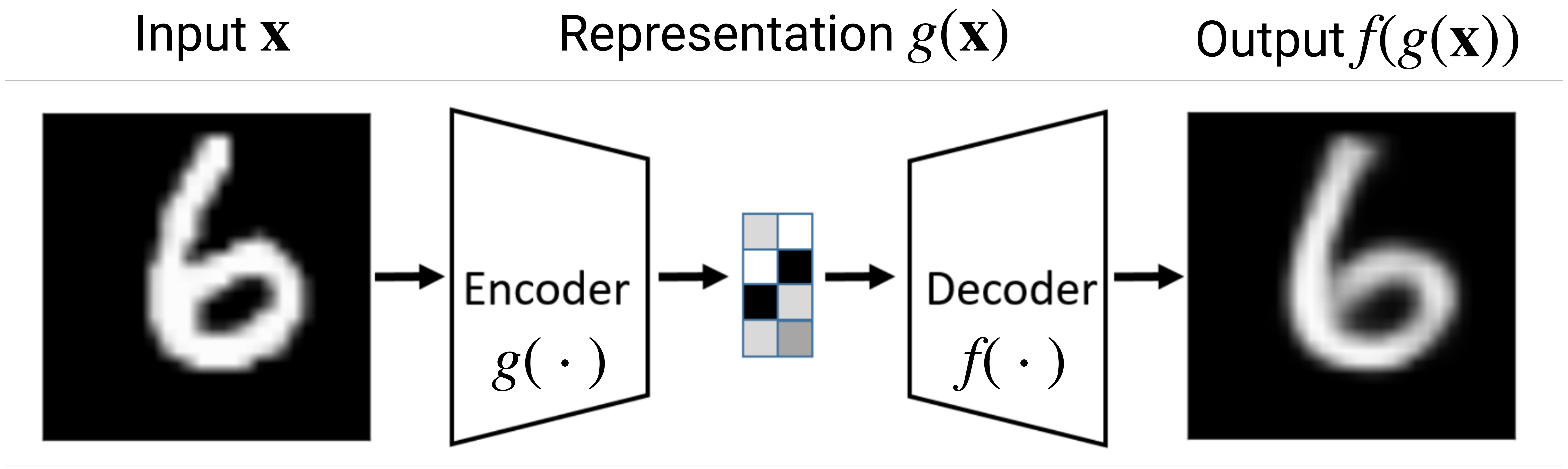
Autoencoders

Autoencoder

- Originally an approach for representation learning
- **Idea.** Train a neural network that can do dimensionality reduction
  - Replace the linear model with neural nets
  - Use SGD to optimize the reconstruction loss

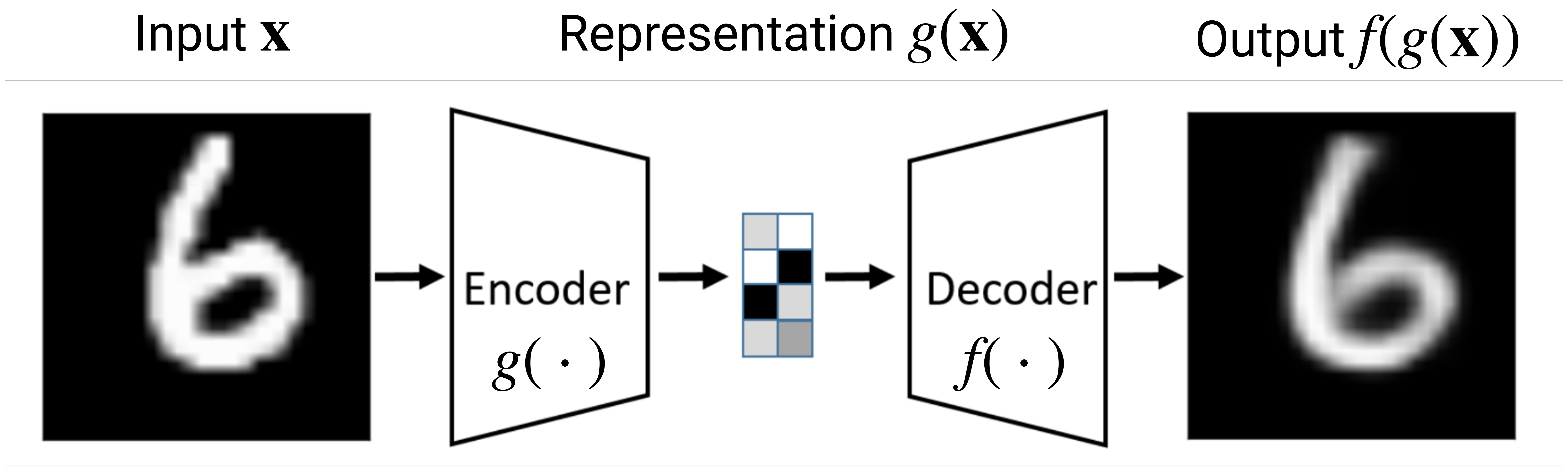


$$\min_{f,g} \mathbb{E}_{\mathbf{x}} \|\mathbf{x} - f(g(\mathbf{x}))\|^2$$



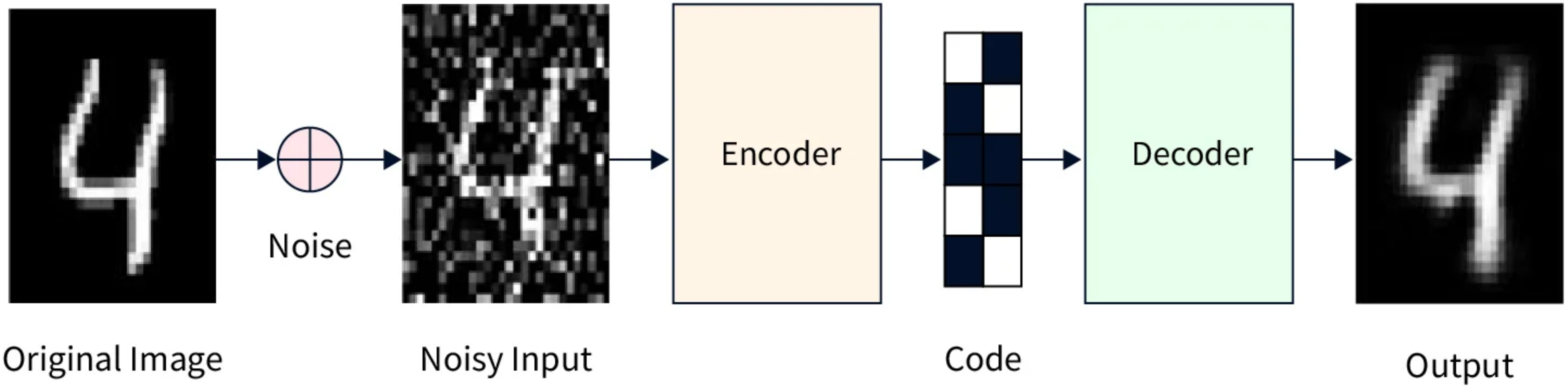
Autoencoder

- **Problem.** As NNs are too powerful, AEs end up learning trivial solutions
  - Overfits to learning an identity function $f(g(x)) = x$, without making the encoder $g(\cdot)$ meaningful
- **Naïve.** Reduce the dimension of $g(\mathbf{x})$ further — extreme hourglass
  - Not very effective in many cases…



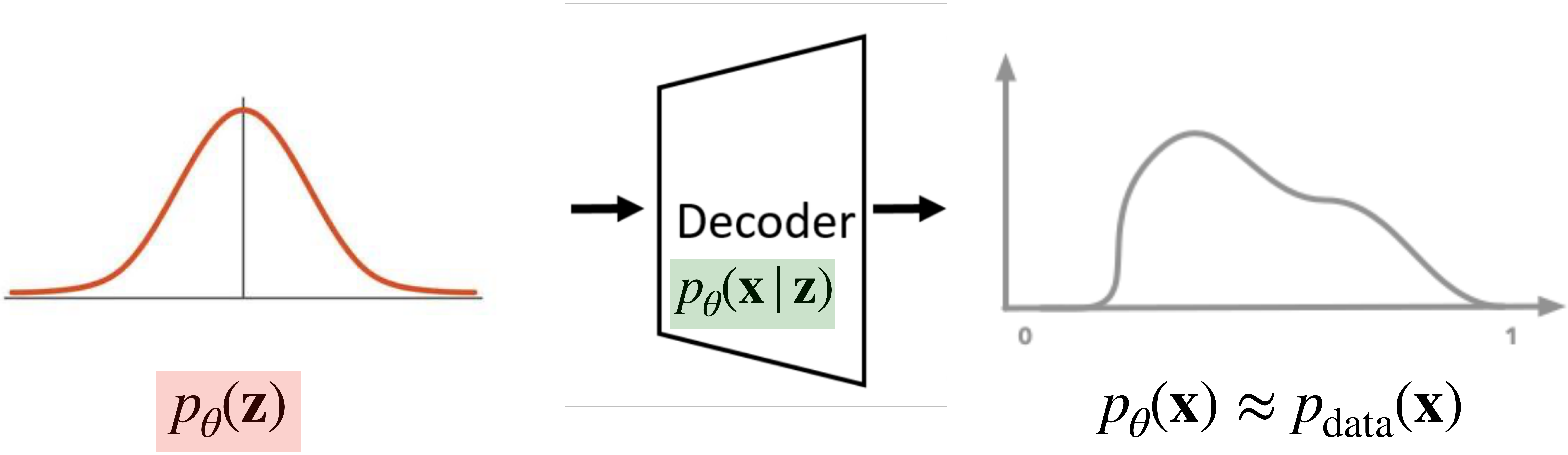
Denoising Autoencoder

- **Idea.** Add noise to the input image, and train to recover the clean image
  - Never solved by identity function
  - Requires understanding our real images look like
    - Tells apart from random noise
- <u>Other examples</u>. Sparse autoencoders…



Variational Autoencoder

- Structurally similar to autoencoders, but very different in goals
- **Goal.** Train a decoder and a distribution such that
  - Input. Send in a distribution $p_\theta(\mathbf{z})$
  - Output. Get a data-generating distribution $p_\theta(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$



Variational Autoencoder

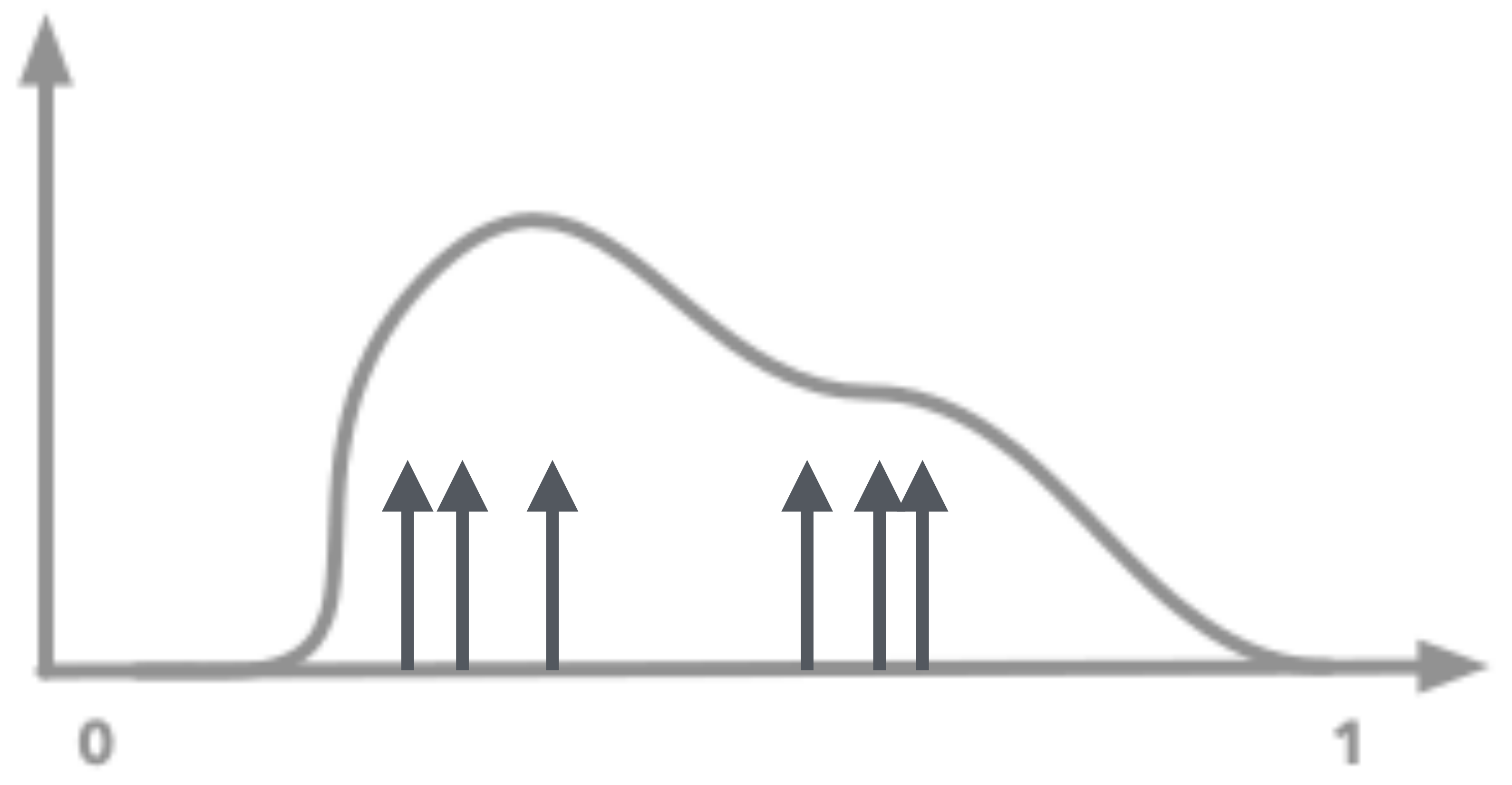
- **Training.** Optimize the log probability

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

- Note. Equivalent to minimizing some distance between the empirical distribution $\hat{p}$, and the modeled distribution p_θ
  - The "Kullback-Leibler divergence"

$$\min_{\theta} D(\hat{p}(\mathbf{x}) \| p_{\theta}(\mathbf{x}))$$

where $D(p \parallel q) = \mathbb{E}_p \log(p/q)$



$$p_{\theta}(\mathbf{x}) \approx \hat{p}(\mathbf{x})$$

Variational Autoencoder

- **Problem.** Computing the marginal distribution is intractible

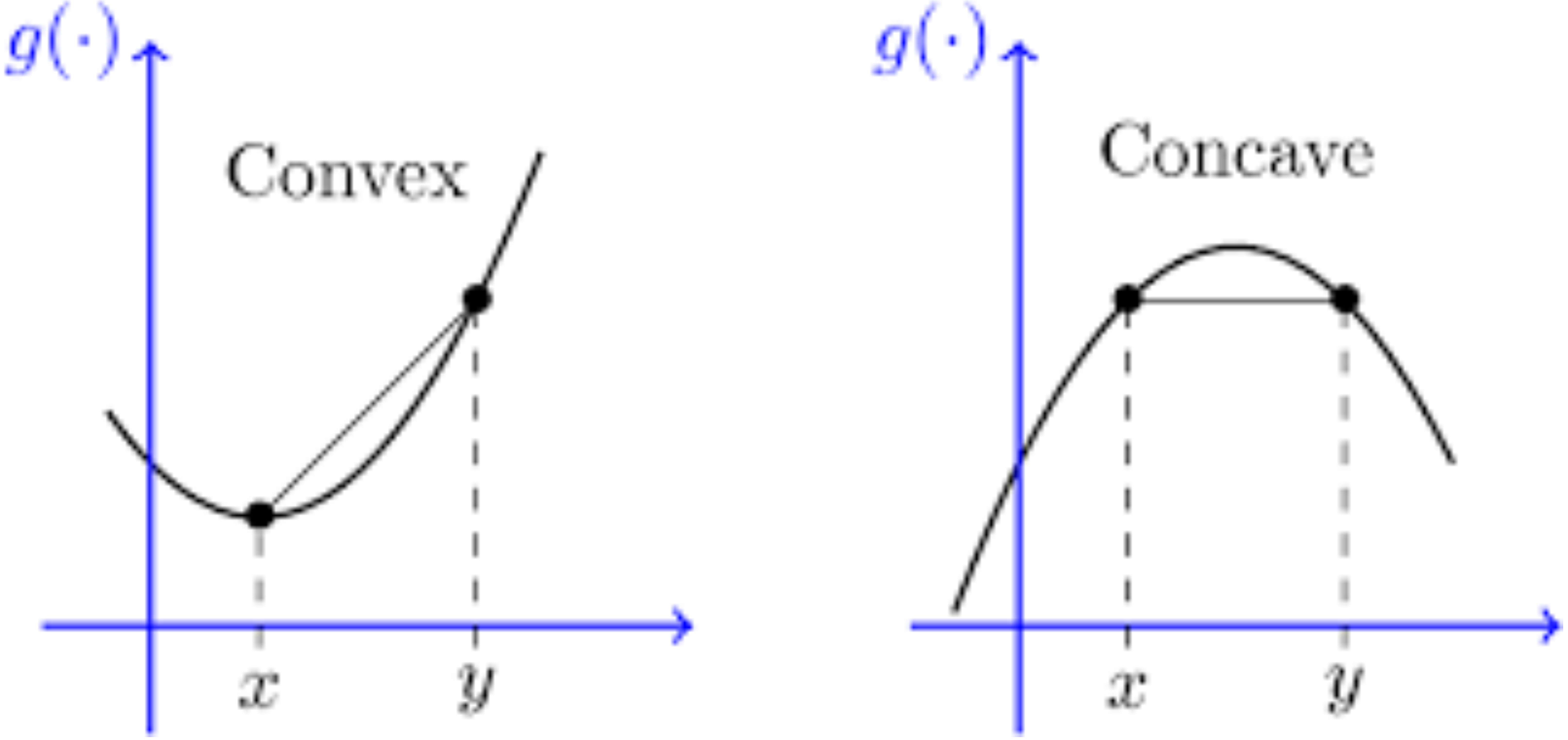
$$p_\theta(\mathbf{x}_i) = \int p_\theta(\mathbf{x}_i \mid \mathbf{z}) p_\theta(\mathbf{z}) \, \mathrm{d}\mathbf{z}$$

  - We can try drawing many samples from $p_\theta(\mathbf{z})$ to approximate this with samples (i.e., Monte Carlo approach)
  - … but this is too costly

- **Idea.** Maximize some lower bound of $\log p_\theta(\mathbf{x})$
  - Evidence lower bound (ELBO)
    (We saw this in expectation-maximization!)

Recap: ELBO

- **Recap: Jensen's inequality.**
  - For concave function $f(\cdot)$, we have $\mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$
  - For convex function $f(\cdot)$, we have $\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$



Recap: ELBO

- For some arbitrary $q_\phi(\mathbf{z})$, we have

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x} \,|\, \mathbf{z})\, \mathrm{d}\mathbf{z} \\
&= \log \int q_\phi(\mathbf{z}) \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z})} p_\theta(\mathbf{x} \,|\, \mathbf{z})\, \mathrm{d}\mathbf{z} \\
&\geq \int q_\phi(\mathbf{z}) \cdot \log \left[\frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z})} p_\theta(\mathbf{x} \,|\, \mathbf{z}) \right] \mathrm{d}\mathbf{z} \\
&= -D(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) + \mathbb{E}_{\mathbf{z} \sim q_\phi}[\log p_\theta(\mathbf{x} \,|\, \mathbf{z})]
\end{aligned}$$

Recap: ELBO

$$\log p_\theta(\mathbf{x}) \geq -D(q_\phi(\mathbf{z}) \| p_\theta(\mathbf{z})) + \mathbb{E}_{\mathbf{z}\sim q_\phi}[\log p_\theta(\mathbf{x} \mid \mathbf{z})]$$

- We can sample from $q_\phi(\mathbf{z})$ and measure the loss
  - We can choose $q_\phi(\mathbf{z})$ dependently on $\mathbf{x}$ – thus $q_\phi(\mathbf{z} \mid \mathbf{x})$
  - Choose the option that maximizes the RHS

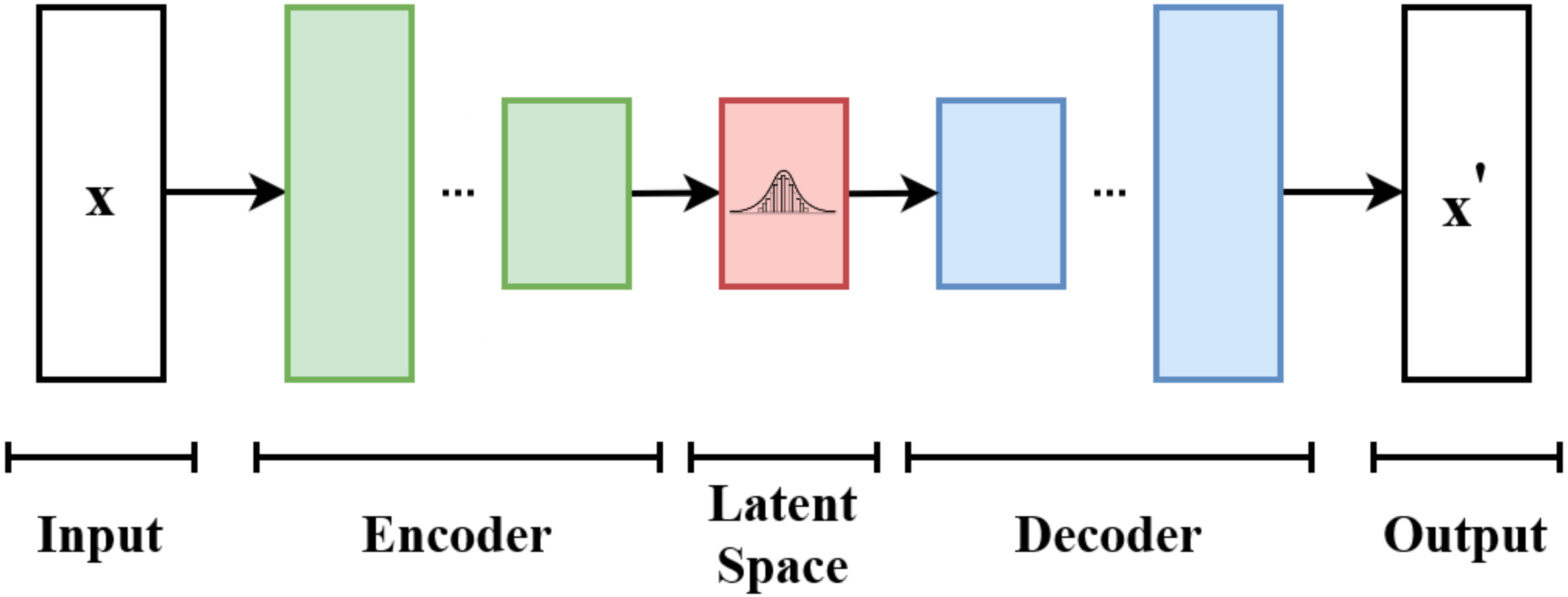
- In other words, we can solve:

$$\max_\theta \log p_\theta(\mathbf{x}_i) \geq \max_\theta \max_\phi \left(-D(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\cdot \mid \mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i \mid \mathbf{z})] \right)$$

  - We train both $q_\phi(\mathbf{z} \mid \mathbf{x})$ and $p_\theta(\mathbf{x} \mid \mathbf{z})$ (and not $p_\theta(\mathbf{z})$)

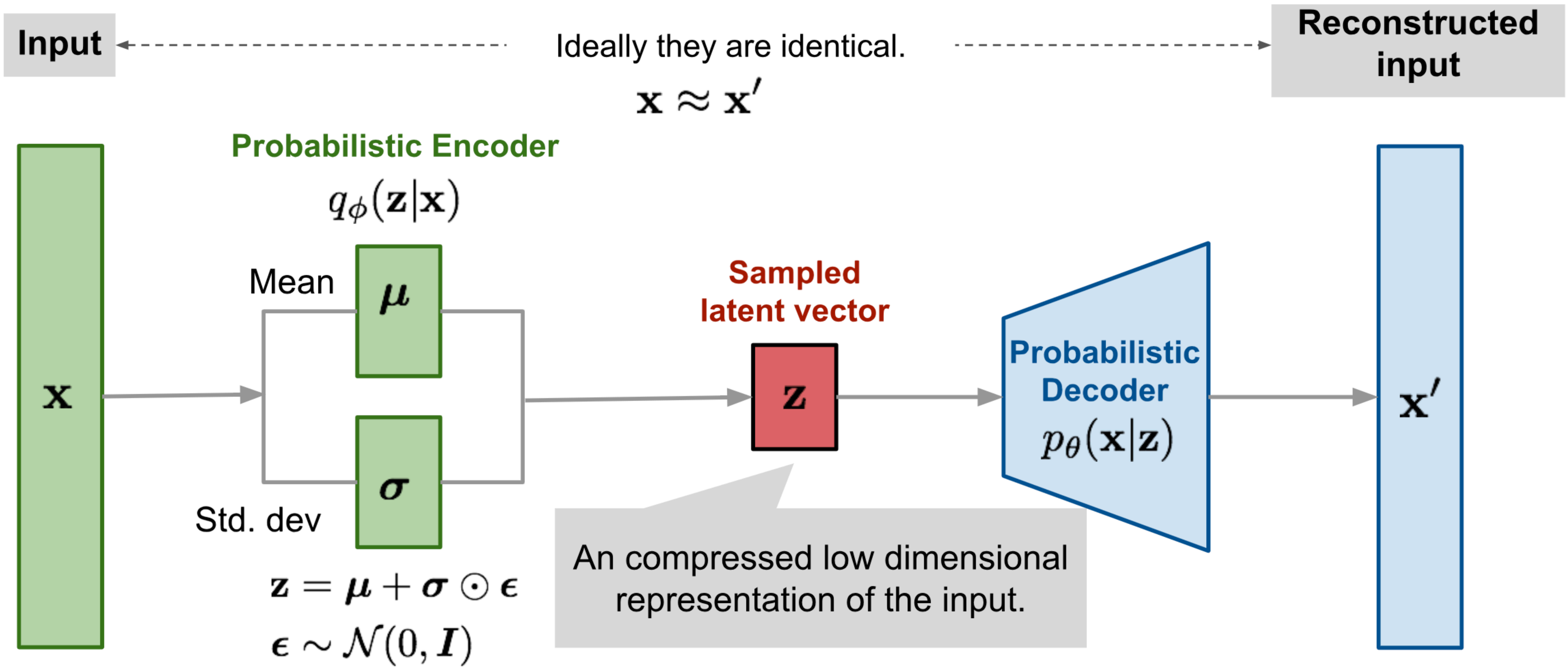
Variational Autoencoders

- **Question.** How do we model $q_\phi(\mathbf{z} \mid \mathbf{x})$?
  - Answer. Jointly train a probabilistic encoder that expresses $q_\phi(\mathbf{z} \mid \mathbf{x})$
  - But how do we implement a probabilistic encoder?



Variational Autoencoders

- **Idea: Reparametrization Trick.**
  Model $q_\phi(\mathbf{z} \mid \mathbf{x})$ as a conditional Gaussian $\mathcal{N}(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}^2)$
  - $\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}$ are the outputs of a neural network encoder



Variational Autoencoders

- Now, look at the optimization problem:

$$\max_{\theta} \max_{\phi} \left(-D(q_\phi(\mathbf{z} \mid \mathbf{x}_i) \| p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\cdot \mid \mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i \mid \mathbf{z})] \right)$$

- First, take a look at the second term
  - By using the model $p_\theta(\mathbf{x}_i \mid \mathbf{z}) = \mathcal{N}(f_\theta(\mathbf{z}), \eta \cdot I_d)$, it becomes

$$-\mathbb{E}_{q_\phi(\cdot \mid \mathbf{x}_i)} \left[\frac{1}{2\eta} \| \mathbf{x}_i - f_\theta(\mathbf{z}_i) \|^2 \right] + \text{const}\,.$$

  - That is, simply use the squared loss
    - A bit more complicated if variances are also trained

Variational Autoencoders

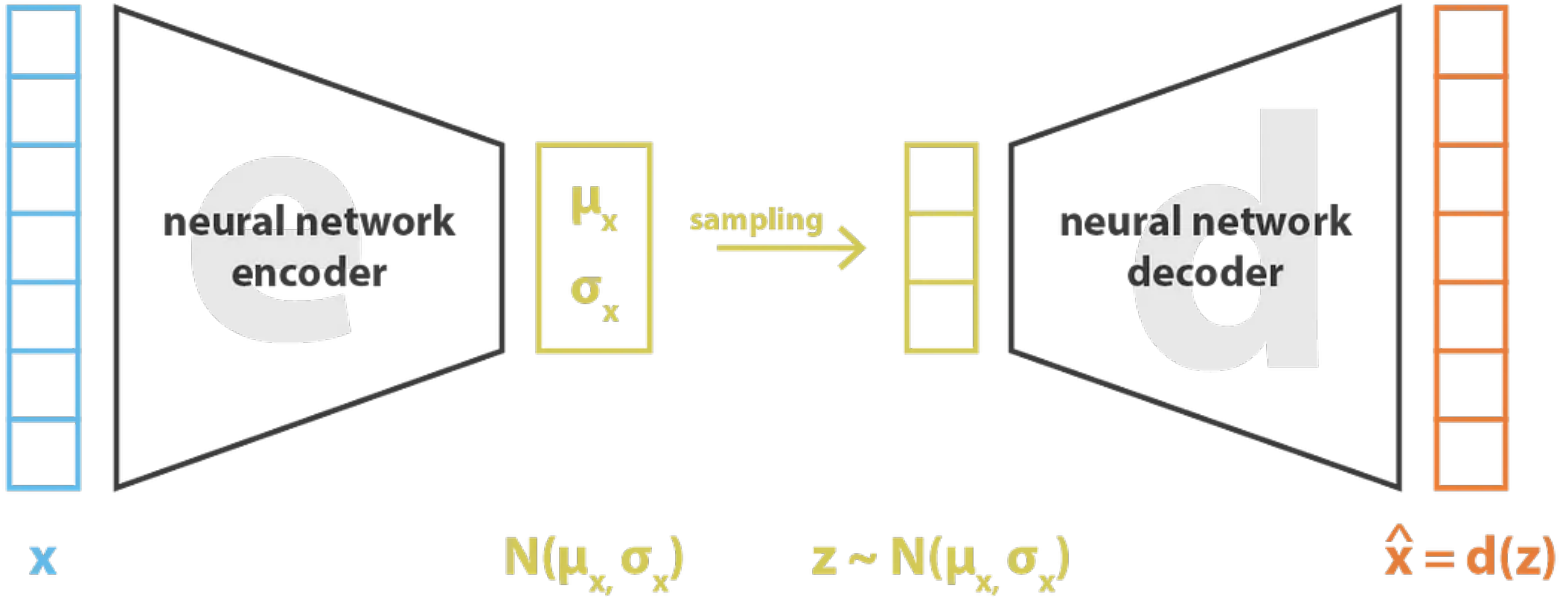
$$\max_{\theta} \max_{\phi} \left(- D(q_\phi(\mathbf{z} \,|\, \mathbf{x}_i) \| p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\cdot|\mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i \,|\, \mathbf{z})] \right)$$

- If we use the Gaussian encoder

$$q_\phi = \mathcal{N}(\mu_{\mathbf{x}_i}, \sigma_{\mathbf{x}_i} \cdot I_k)$$

then this simply becomes ℓ^2 regularizers on μ, σ

  - Check by yourself

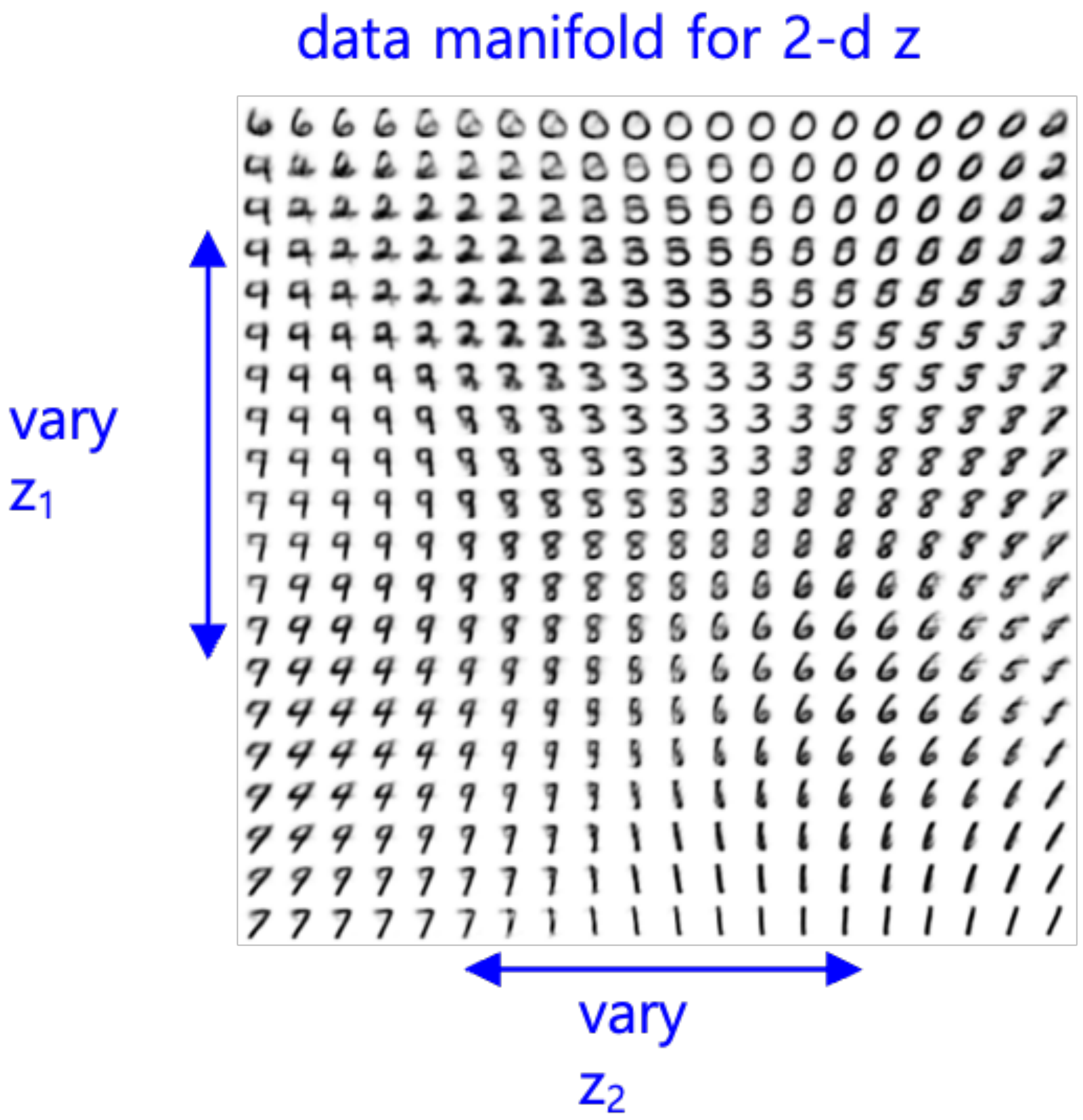


- Thus, a squared loss & ℓ^2 regularizer

$$\text{loss} = \| x - \hat{x} \|^2 + \text{KL}[\, N(\mu_x, \sigma_x), N(0, I) \,] = \| x - d(z) \|^2 + \text{KL}[\, N(\mu_x, \sigma_x), N(0, I) \,]$$

Properties

- **Advantages.** Known to enjoy nice disentanglements
  - Each dimension of $\mathbf{z}$ correspond to clear semantic concepts

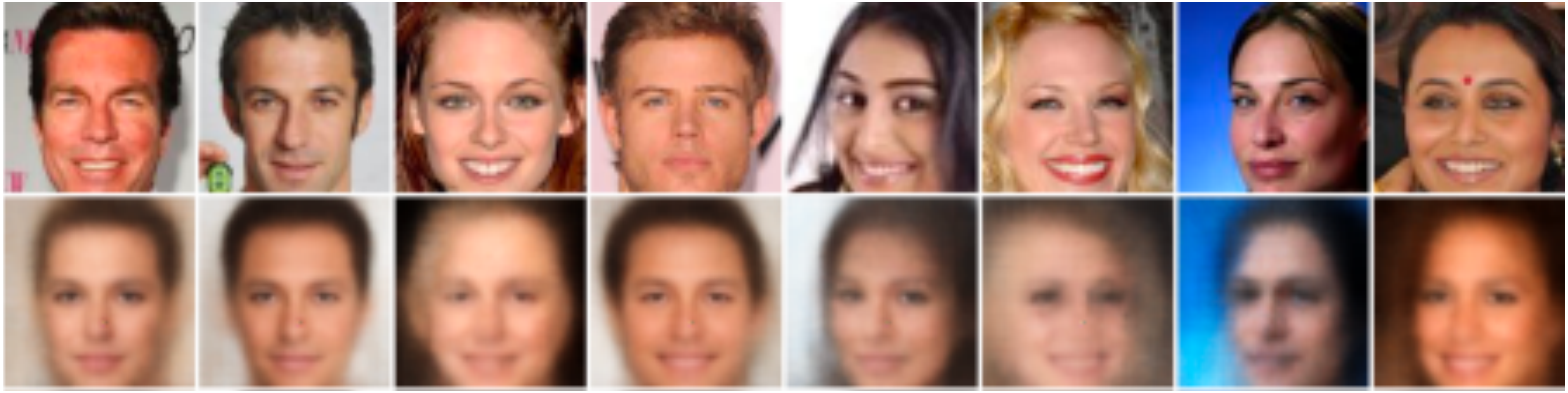


Properties

- **Advantages.** Known to enjoy nice disentanglements
  - Each dimension of $\mathbf{z}$ correspond to clear semantic concepts

Properties

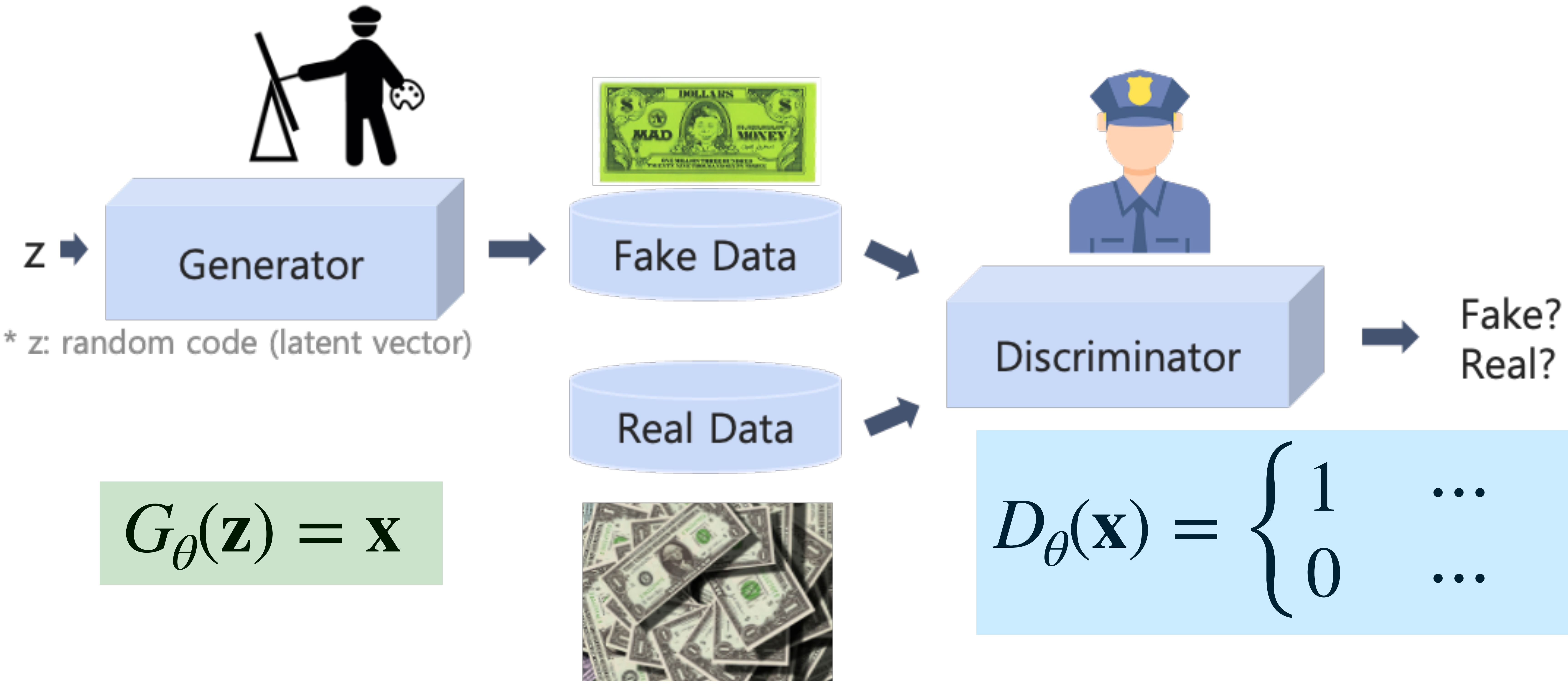
- **Limitations.** Generate rather blurry images
  - Clearly distinguishable from real images
    (at least back then… as technologies advance fast)



Generative Adversarial Nets

Generative Adversarial Nets

- **Idea.** Train explicitly for the "hard-to-distinguish" propety
  - View generative process as a two-player game
  - Discriminator. Tries to distinguish the real / fake
  - Generator. Tries to fool the discriminator



$$G_\theta(\mathbf{z}) = \mathbf{x}$$

$$D_\theta(\mathbf{x}) = \begin{cases} 1 & \cdots \quad \text{if fake} \\ 0 & \cdots \quad \text{if real} \end{cases}$$

Generative Adversarial Nets

- **Training.** Jointly train the generator and the discriminator
  - Solve the minimax optimization

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x}\sim\hat{p}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p(z)} \log(1 - D_{\theta_d} \circ G_{\theta_g}(z))\right]$$

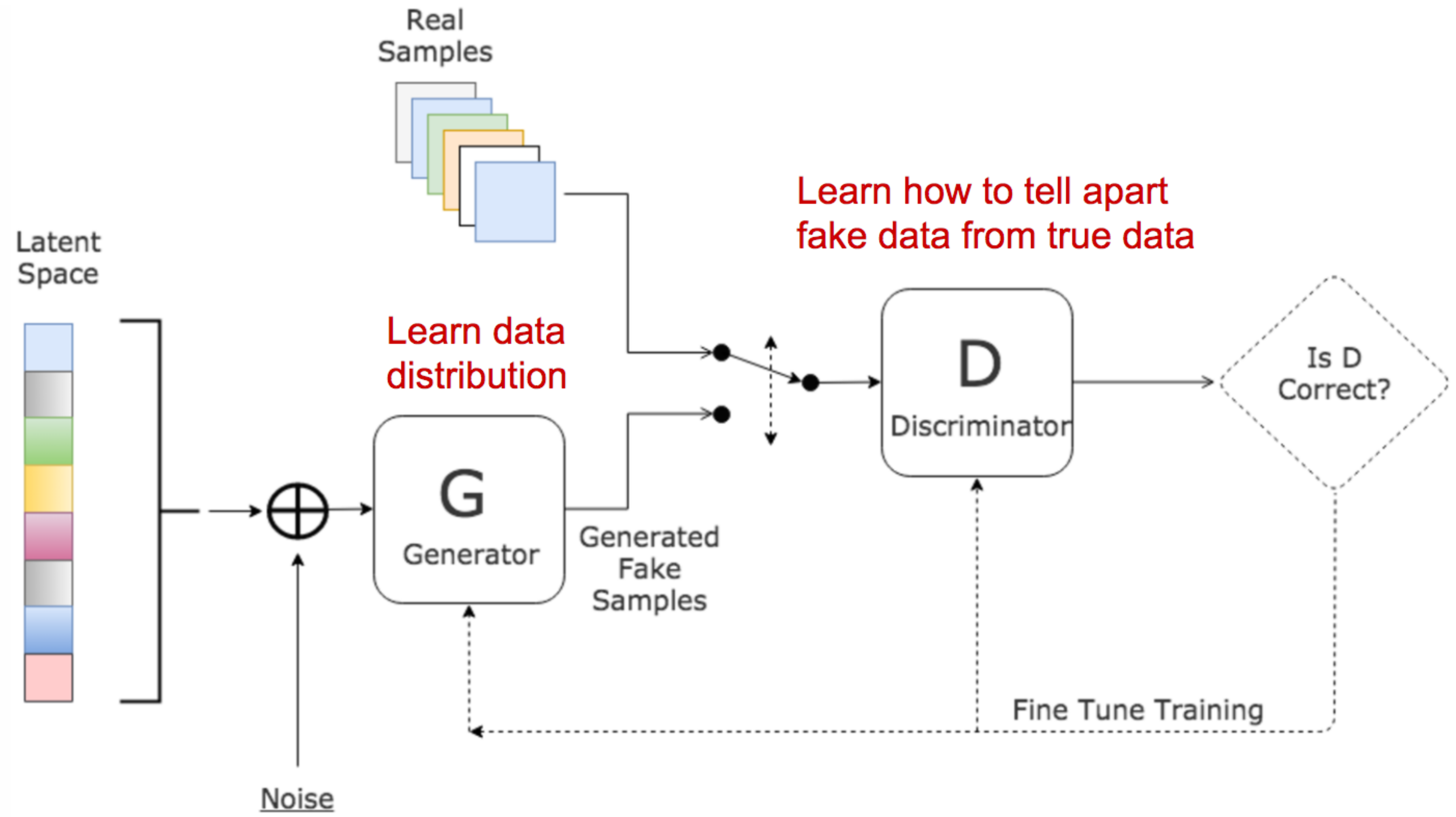
Discriminator declares real image to be real

Discriminator declares fake image to be fake

  - Discriminator outputs the likelihood of being real: $D_{\theta_d}(\mathbf{x}) \in [0,1]$
  - Note. Equivalent to the Jensen-Shannon divergence

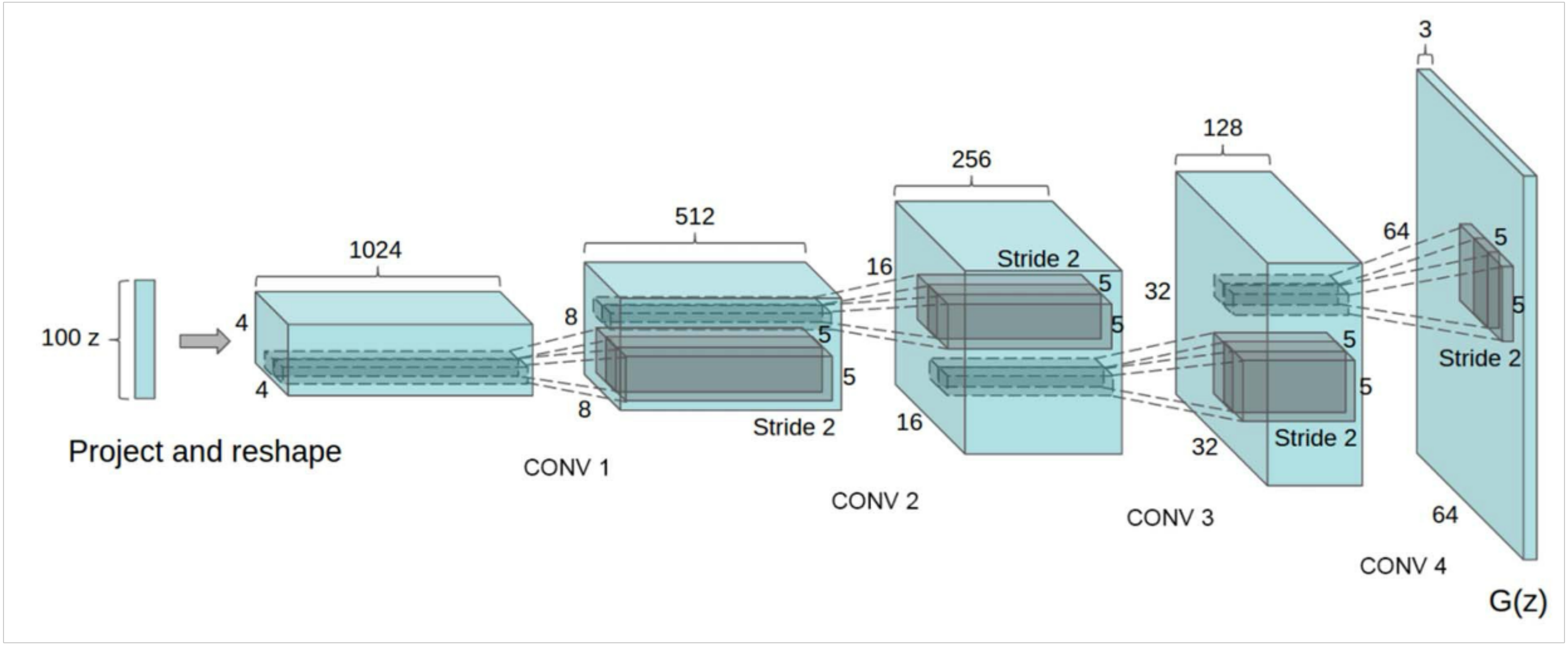
$$D\left(p_\theta \middle\| \frac{\hat{p} + p_\theta}{2}\right) + D\left(\hat{p} \middle\| \frac{\hat{p} + p_\theta}{2}\right)$$

Generative Adversarial Nets



Generative Adversarial Nets

- **Architecture.** Generator uses convolutional layers as well:

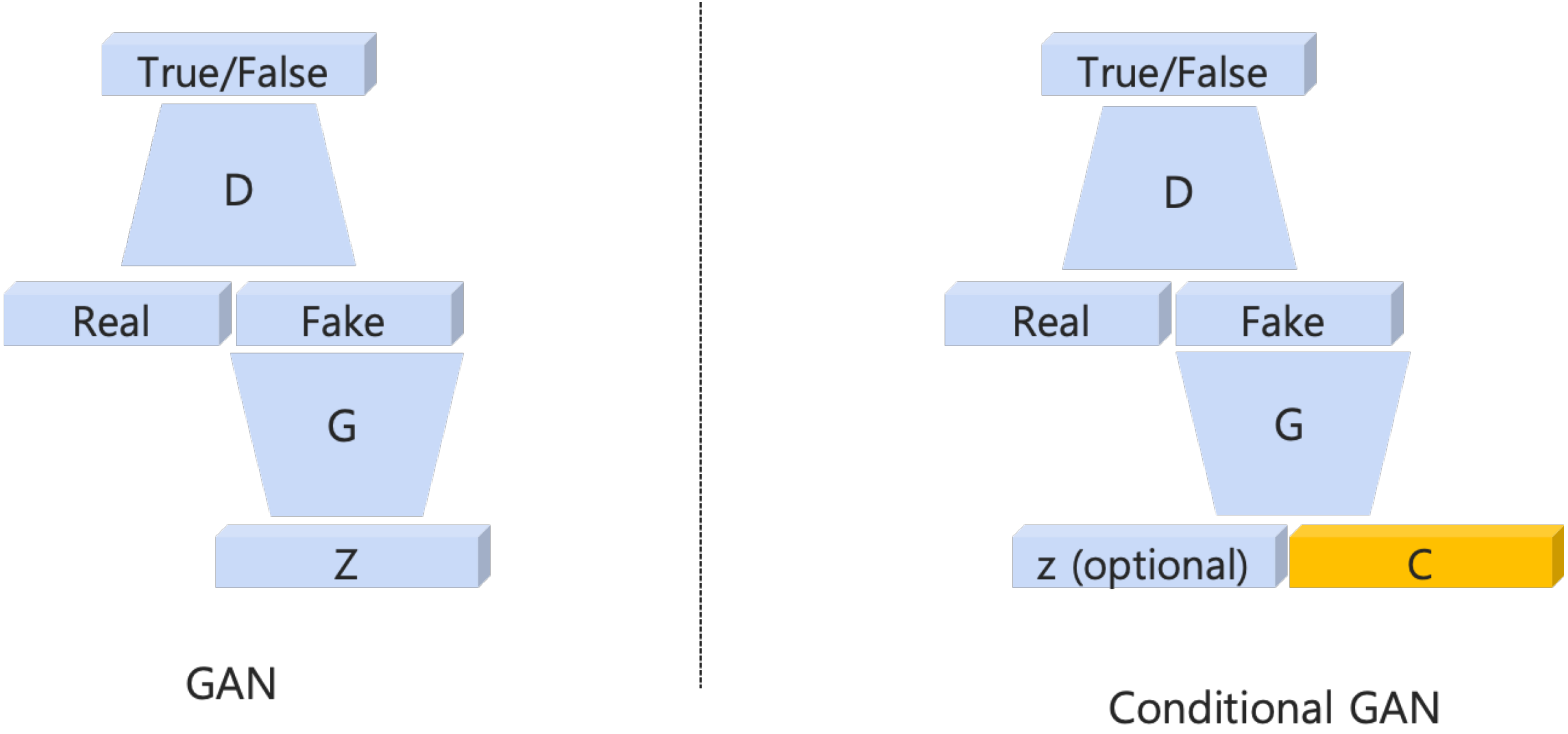


Generative Adversarial Nets

- **Advantages.** GAN can give very sharp images

Conditional GAN

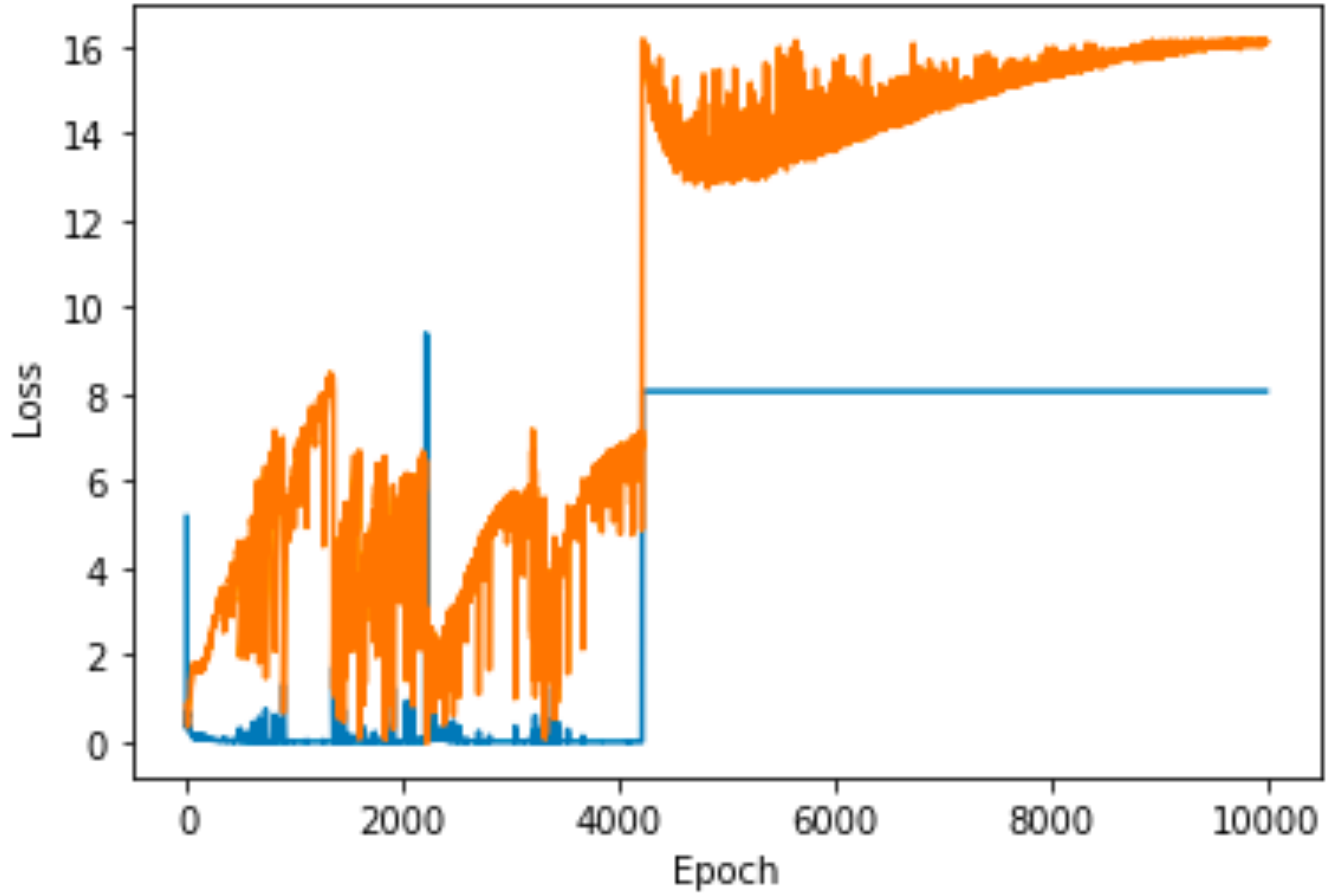
- **Idea.** Add class / text information to the latent code
  - Generate realistic images under specific conditions



Conditional GAN

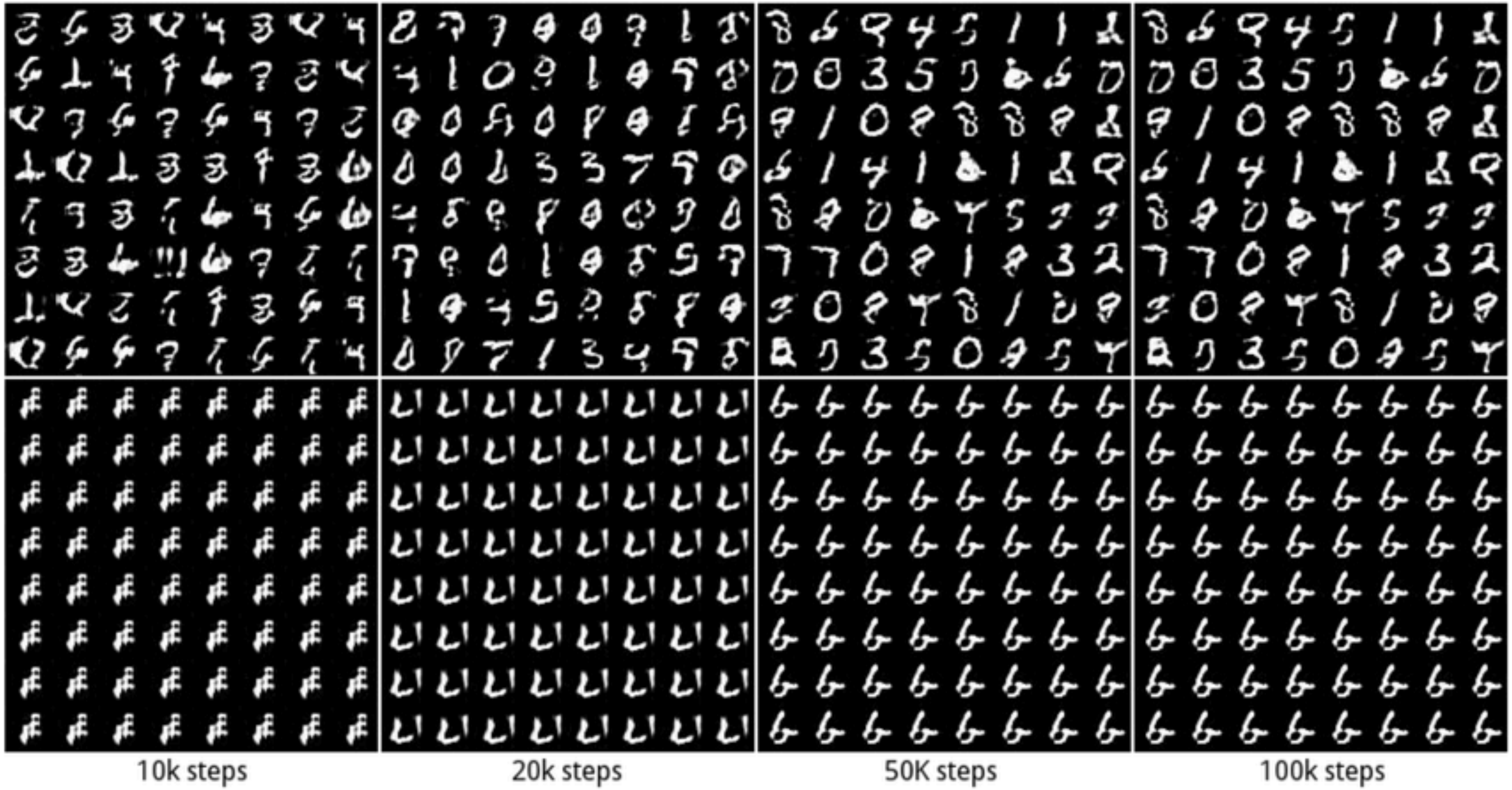
Limitations

- GAN training is known to be very unstable
  - If the discriminator works too well, the generator gives up learning
  - If the generator works too well, the discriminator cannot find any meaningful patterns



Limitations

- As a result, generators tend to overfit to few good solutions
  - called “mode collapse”



Next class

- Diffusion models

</lecture 17>