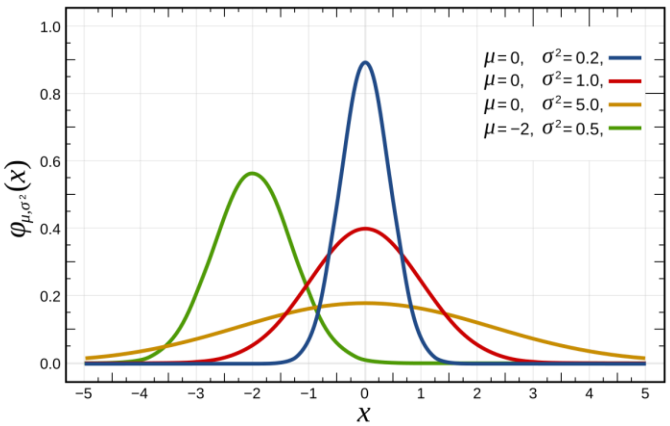
Advanced Machine Learning
Generative Model

Yu Wang
Assistant Professor
Department of Computer Science
University of Oregon

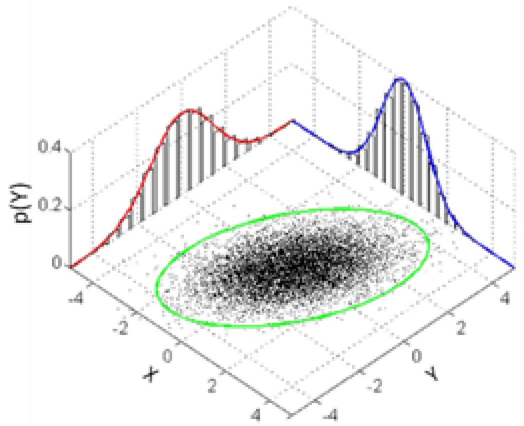
Summary

1D Gaussian Distribution

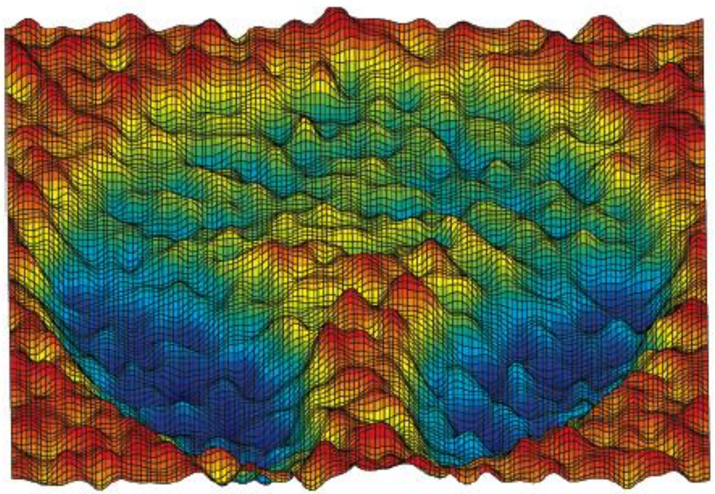


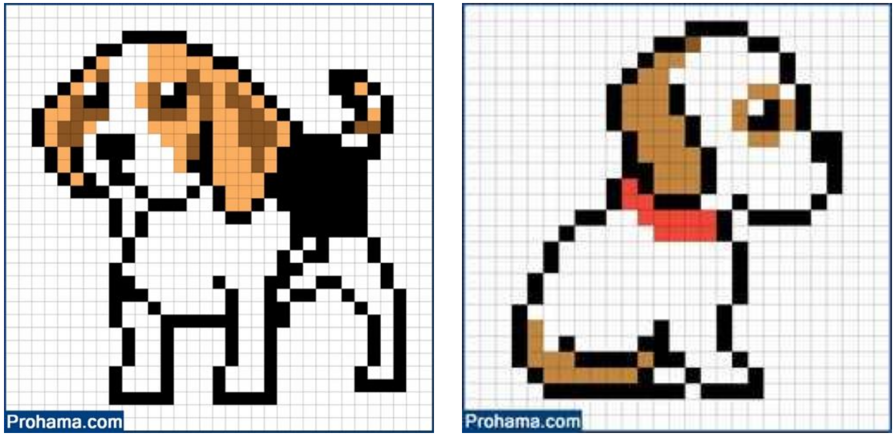
$\mathbb{R}$

2D Gaussian Distribution



$\mathbb{R}^2$





$\mathbb{R}^{256\times256}$

Summary

Probability distribution of the **objective** based on the **observed data**

$$\{x_i\}_{i=1}^{N} \xrightarrow[\text{Good Model}]{} P(x) \xrightarrow[\text{Good Data}]{} x$$

- **Machine Learning Methods**
  - Gaussian Kernel Density Estimation
  - Gaussian Mixture Models

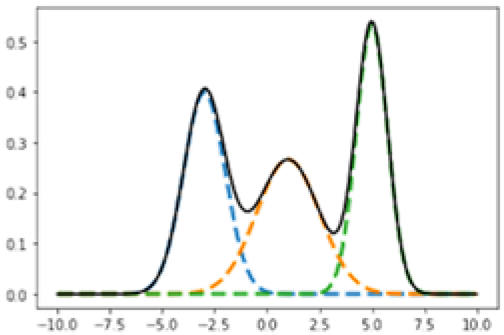
Using existing function to estimate what you do not know that can best fit your observation

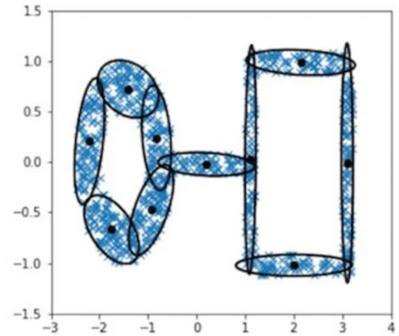
- **Deep Learning Methods**
  - Auto-Encoder (AE)
  - Variational AE (LLM is actually a VAE)
  - Generative Adversarial Network
  - Diffusion Model

Using learnable function to estimate what you do not know that can best fit your observation

Problem?

Using existing function to estimate what you do not know that can best fit your observation





$\mathbb{R}^1, \mathbb{R}^2$

$\downarrow$

$\mathbb{R}^{256\times256}$

What you have is some low-dimensional data
But what you want to model is some high-dimensional data, how it could be?

Problem?

What we want: model any data distribution

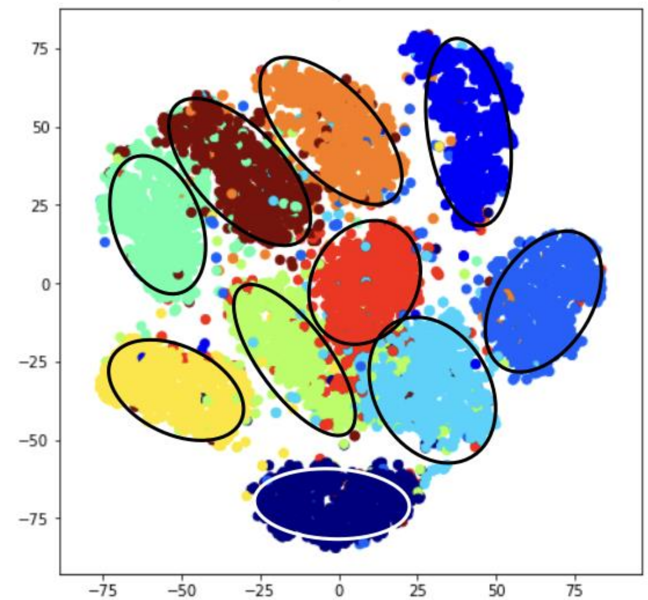
How to transform any data distribution to low dimensional data?

What we have: kernel density estimation to estimate low dimensional PDF

Someway to transform

Transform back

Kernel Density Estimation



Summary

Probability distribution of the **objective** based on the **observed data**

$\{x_i\}_{i=1}^{N} \xrightarrow{\text{Good Model}} P(x) \xrightarrow{\text{Good Data}} x$

- **Machine Learning Methods**
  - Gaussian Kernel Density Estimation
  - Gaussian Mixture Models

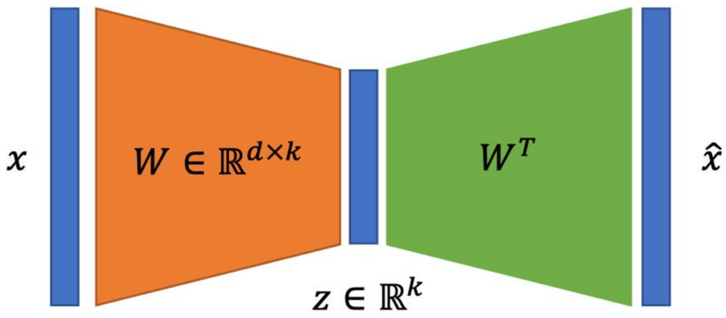
Using existing function to estimate what you do not know that can best fit your observation

PCA Dimensional Reduction

- **Deep Learning Methods**
  - Auto-Encoder (AE)
  - Variational AE (LLM is actually a VAE)
  - Generative Adversarial Network
  - Diffusion Model

Using learnable function to estimate what you do not know that can best fit your observation

From PCA to Auto-Encoder



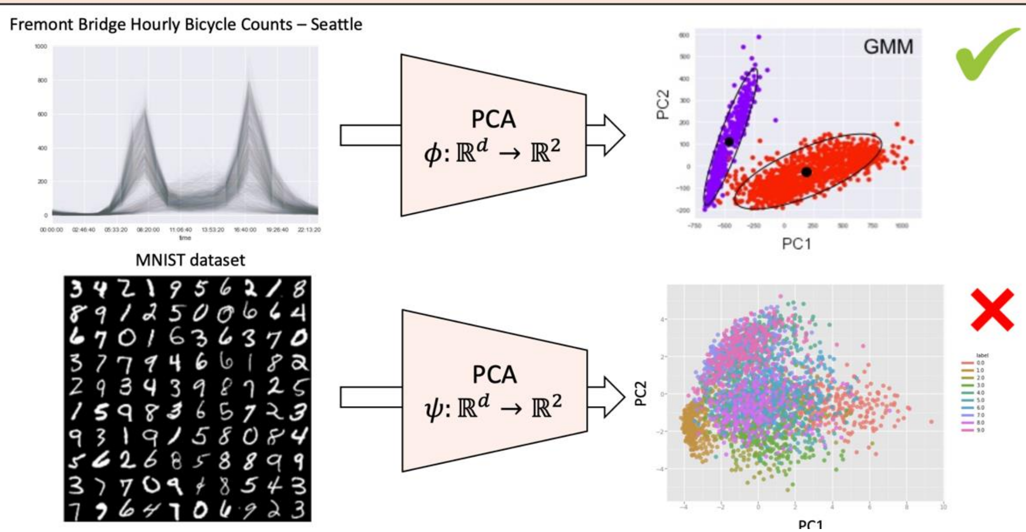
PCA:

- Forward transform: $z = W^T x$
- Inverse transform: $\hat{x} = Wz$

Linear dimensionality Reduction

$$\min_W \mathbb{E}_x[\|x - \hat{x}\|^2] = \mathbb{E}_x[\| x - WW^T x\|^2]$$
$$s.t. \quad W^T W = I_{k\times k}$$

High-dimensional data often lives on non-linear manifolds that cannot be captured by linear models such as PCA

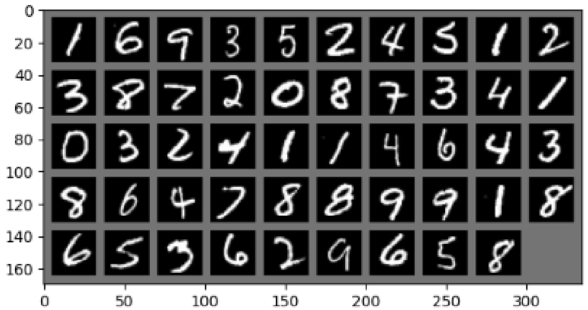


Can we add nonlinearity?

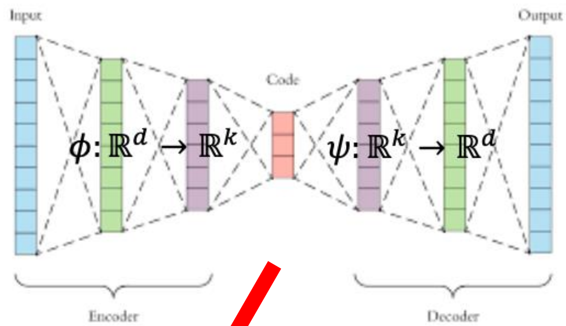
Yes, then it becomes neural network!

Auto-Encoder

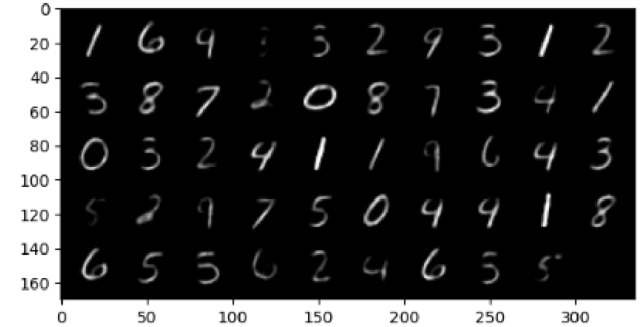
Input

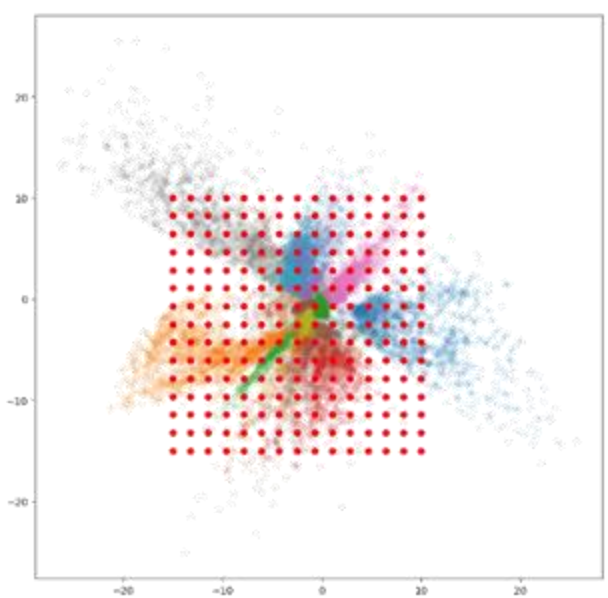


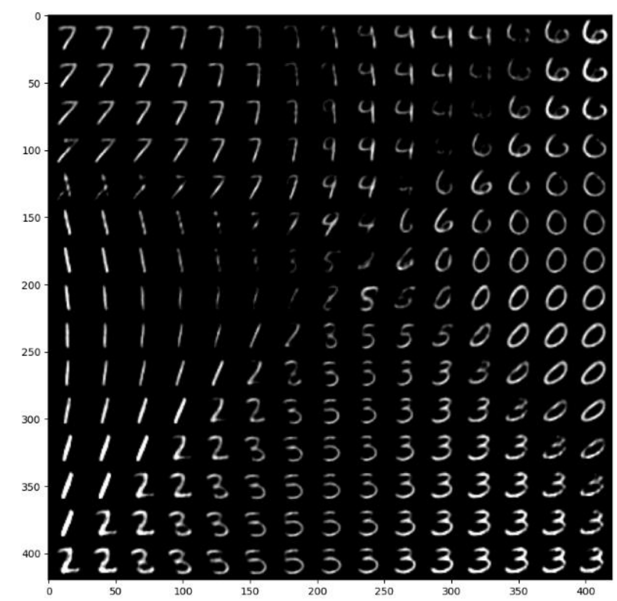
AE



Output

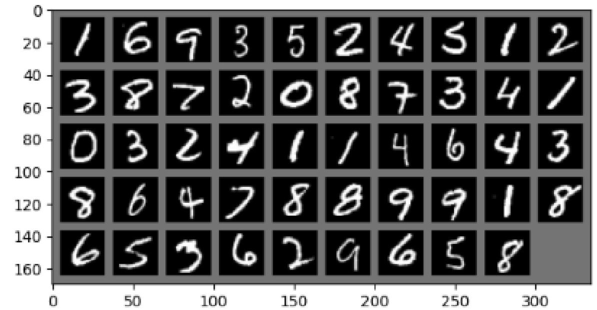




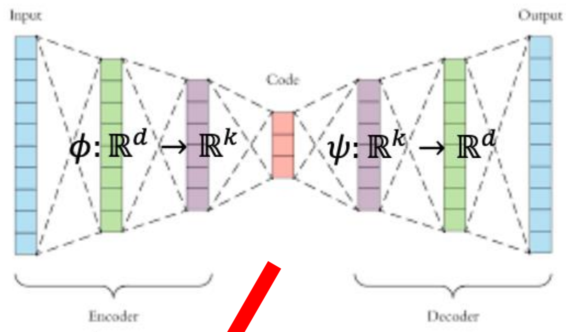


Class-supervised Auto-Encoder

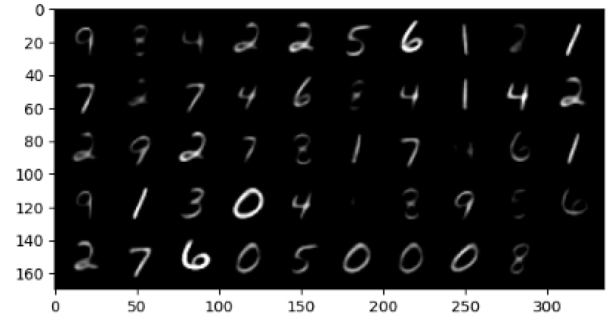
Input

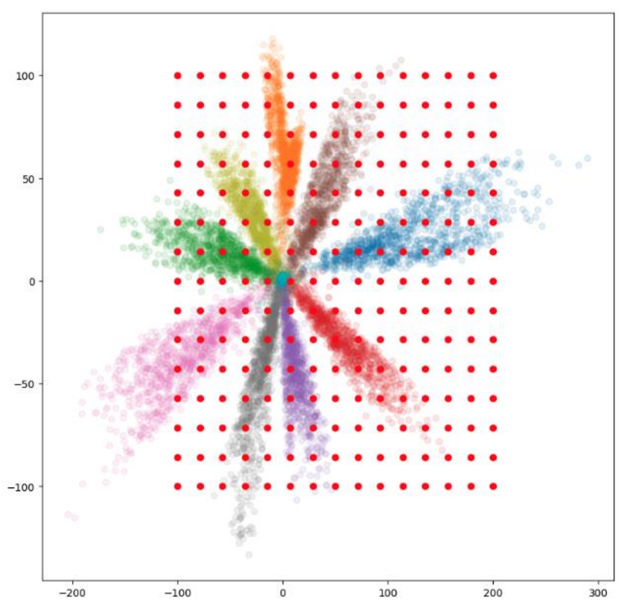


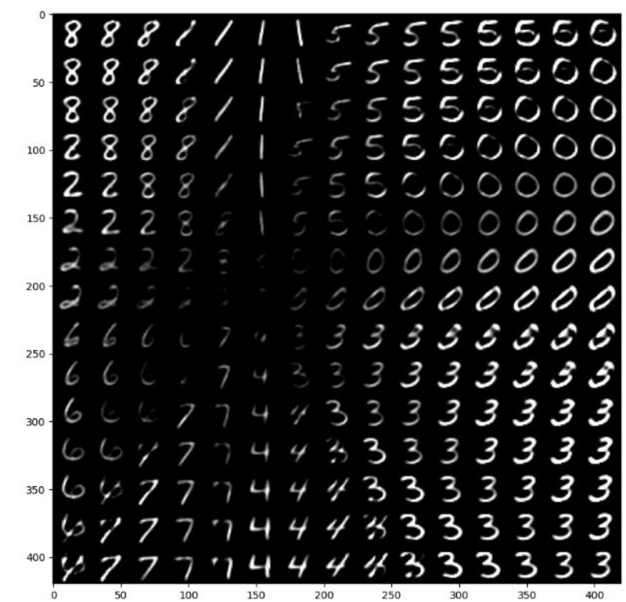
AE



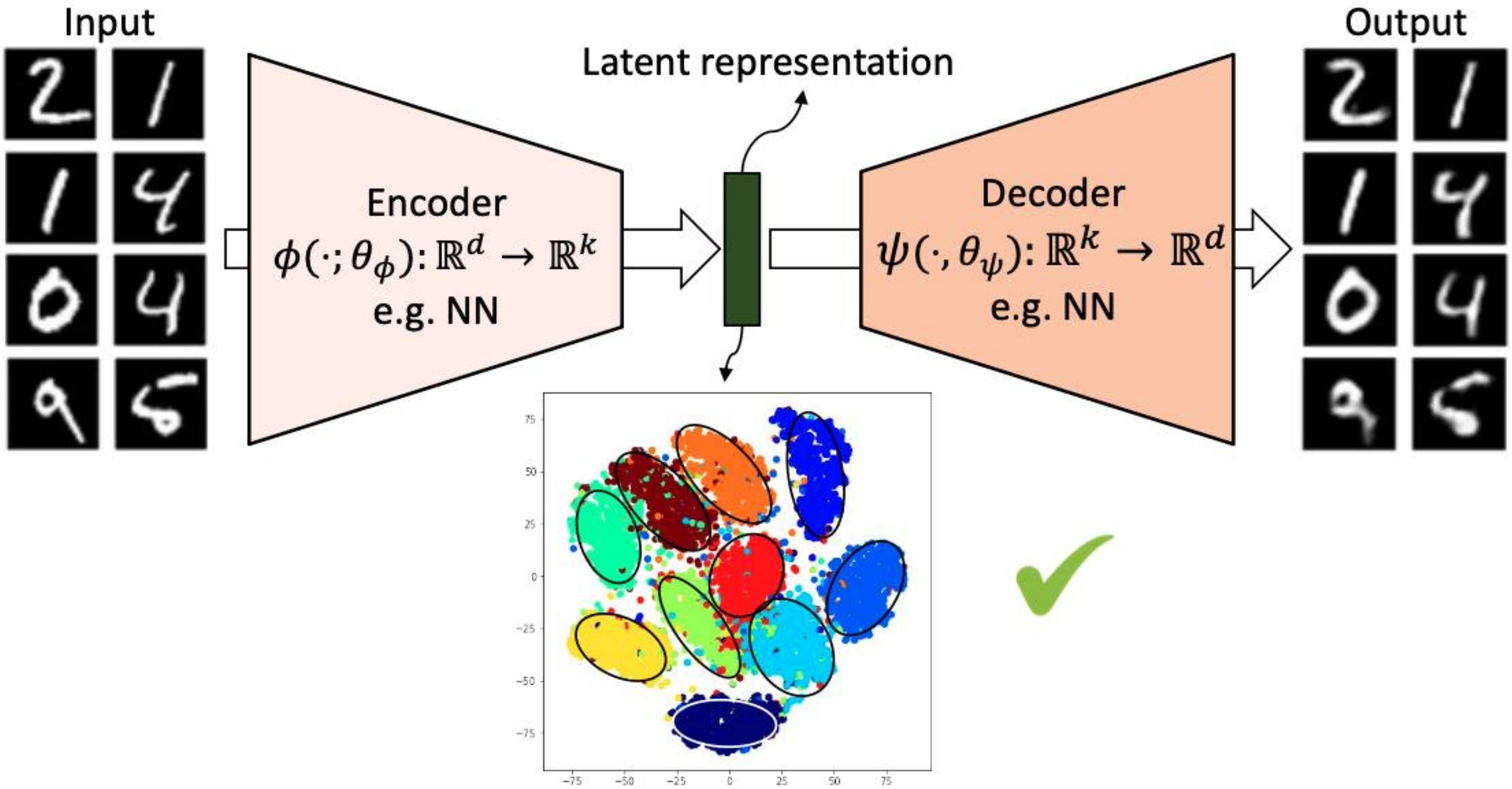
Output







Problem with AE

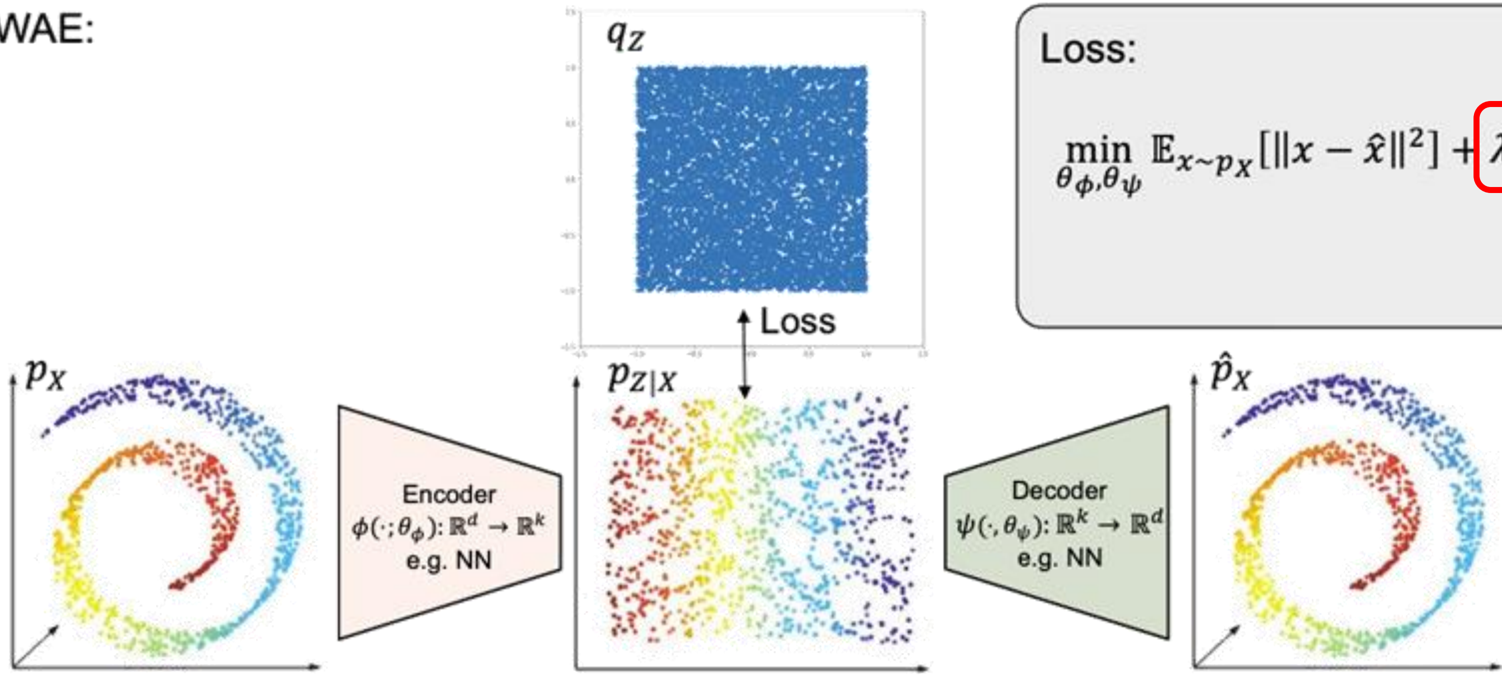


Need to estimate the latent distribution post-hoc!

Solution – Sliced Wasserstein AE

Sliced Wasserstein Distance between two distributions!

SWAE:



Loss:

$$\min_{\theta_\phi,\theta_\psi} \mathbb{E}_{x\sim p_X}[\|x-\hat{x}\|^2] + \boxed{\lambda\, SW(p_{Z|X}, q_Z)}$$

Solution – VAE

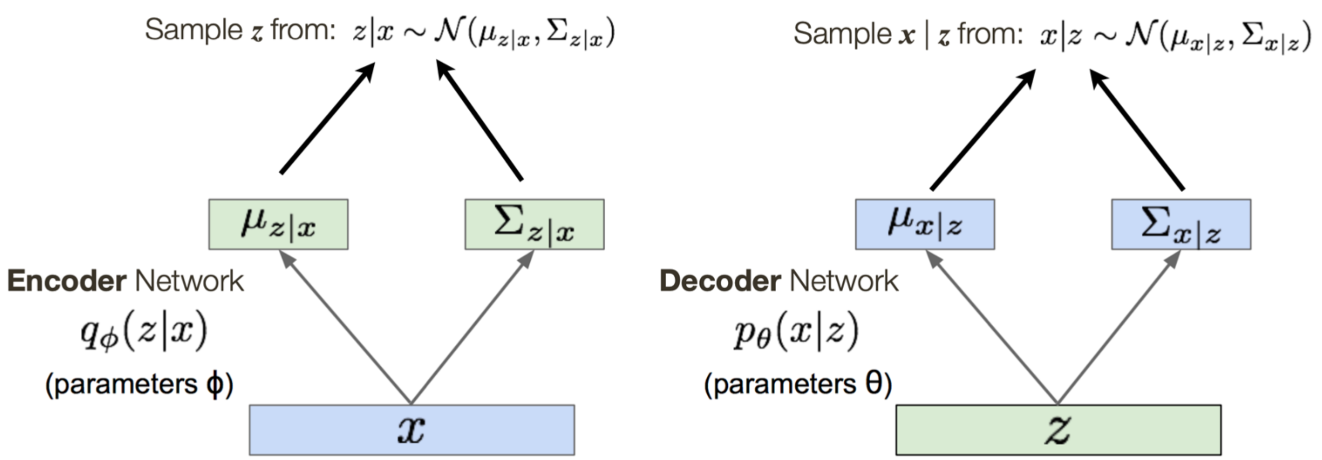
$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

$$\mathbb{E}_{x \sim p_X}[\|x - \hat{x}\|^2]$$

$$D_{\mathrm{KL}}(q_\phi(z|x)\,||\,p(z)) = \frac{1}{2}\sum_{j=1}^{d}\left[\sigma_j^2 + \mu_j^2 - 1 - \log \sigma_j^2\right]$$

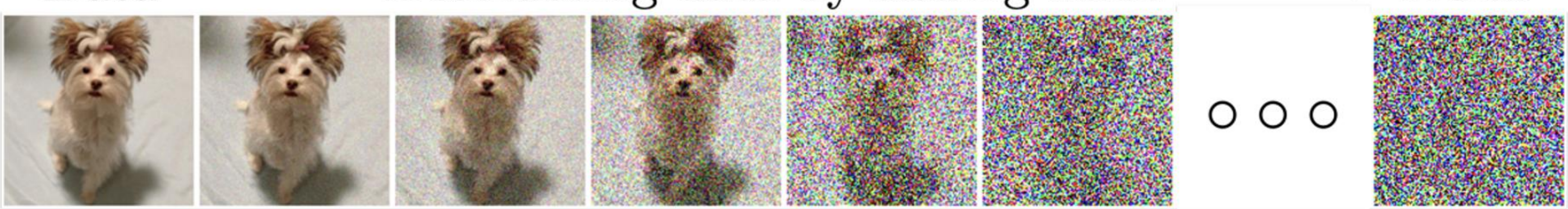
(1) Reconstruction loss: given z – decoder – x and setup the reconstruction loss

(2) KL divergence: how to optimize the KL divergence between two gaussian distributions?



Multi-Step VAE - Diffusion

Data ——— Destructing data by adding noise ⟶ Noise



data distribution $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, $\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_T$ with transition kernel $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$

$\beta_t \in (0, 1)$ is a hyperparameter

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}).$$

with $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$,

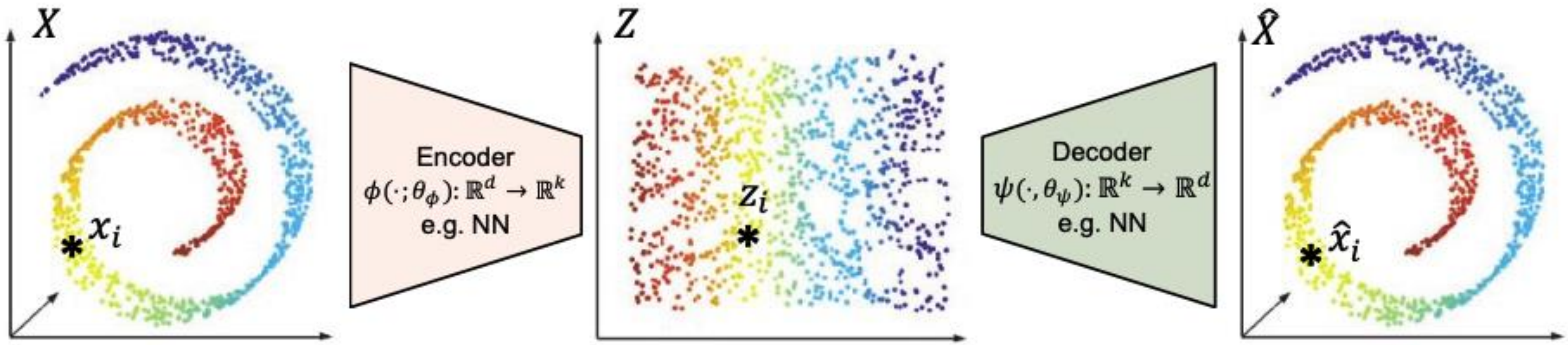
$\boldsymbol{t \to \infty, \alpha_t \to 0,}$

$p(\mathbf{x}_{t-1}|\mathbf{x}_t) \longleftarrow$ $\boldsymbol{q(\mathbf{x}_t|\mathbf{x}_0) \to \mathcal{N}(0, 1)}$

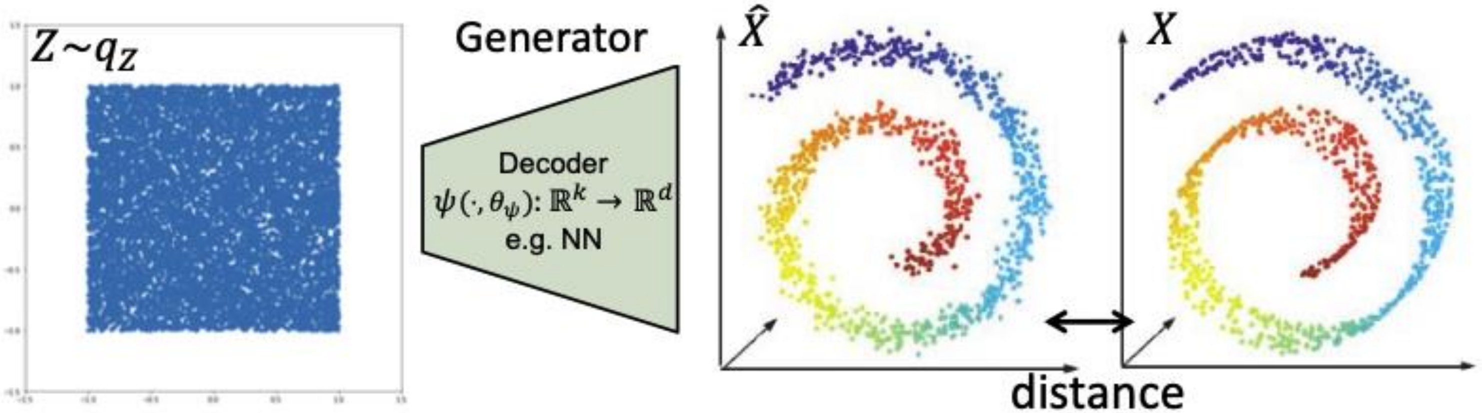
Data ⟵ Generating samples by denoising ——— Noise

Encoder-Less Generation

Auto-encoder based generative modeling

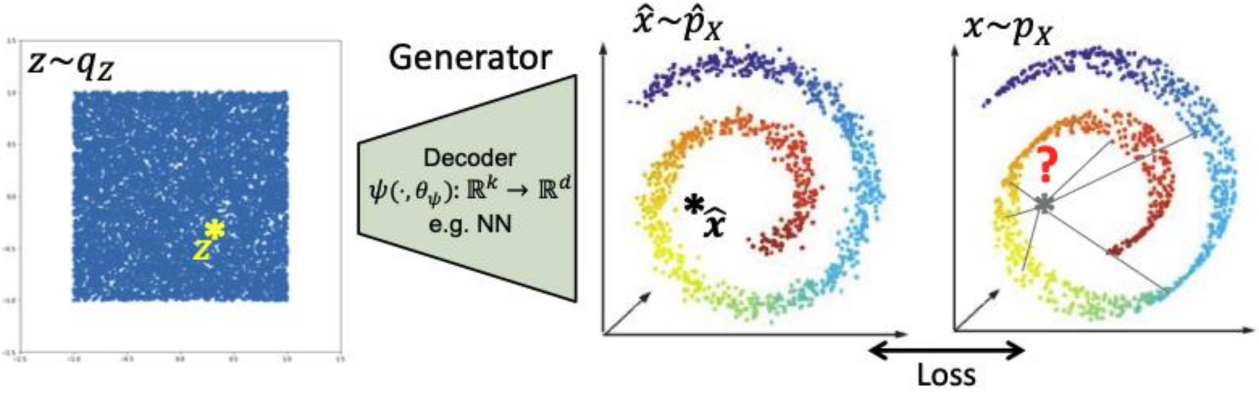


Encoder-less Generative Modeling



Encoder-Less Generation

Generative Adversarial Networks: Main Idea



We sample $z \sim q_Z$ and generate $\hat{x} = \psi(z; \theta_\psi)$. Note that we want $\hat{x}$ to sit on the manifold of the original data, but we do not know the corresponding point for $\hat{x}$ in X!

How to compute the likelihood?

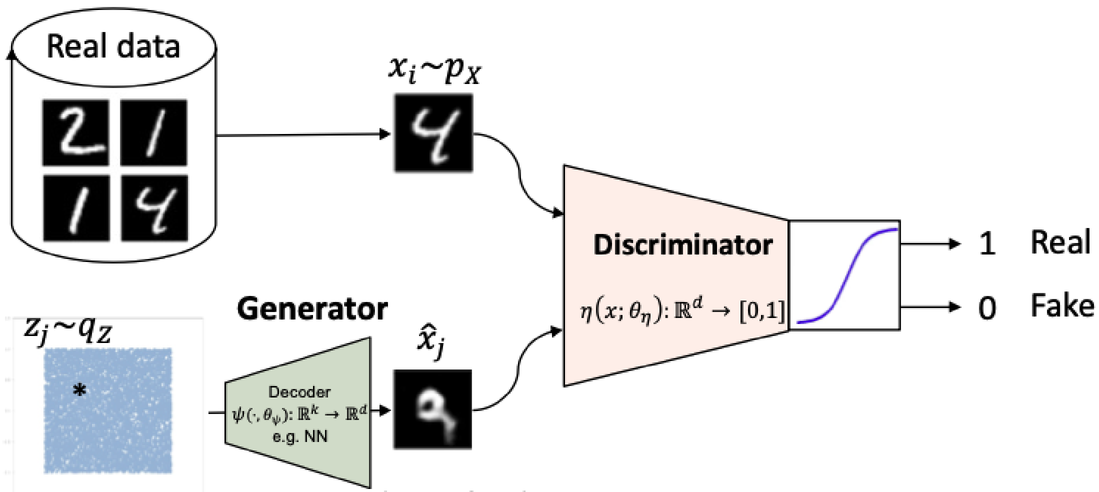
Sample-wise distances won't work in this setting, and one must rely on distribution-based distances.

Can we train a neural network to tell the difference and try to minimize that difference?

Generative Adversarial Networks (GAN)

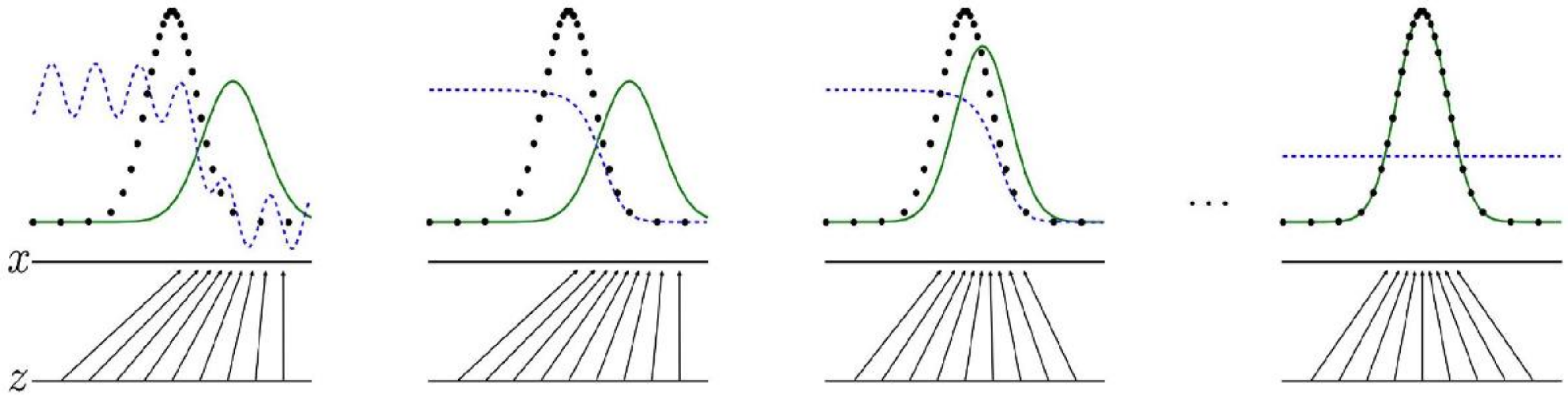
The idea in GANs is to use an adversarial network that tries to distinguish the points on the manifold of the data from any other points in $\mathbb{R}^d$. Let $\eta(x;\theta_\eta): \mathbb{R}^d \to [0,1]$ be the discriminator:

$$\min_{\theta_\psi} \max_{\theta_\eta} \frac{1}{N}\sum_i \log(\eta(x_i;\theta_\eta)) + \frac{1}{M}\sum_j \log\left(1-\eta(\hat{x}_j;\theta_\eta)\right), \qquad where \;\; \hat{x}_j = \psi(z_j;\theta_\psi)$$



$$\min_\psi \max_\eta \mathbb{E}_{p_X}\left[\log(\eta(x))\right] + \mathbb{E}_{q_Z}\left[log(1-\eta(\psi(z)))\right]$$

Generative Adversarial Networks



Ground-truth datapoints

Generative Data Distributions

Discriminator Confidence

GAN's formulation

$$\min_G \max_D V(D, G)$$

- It is formulated as a **minimax game**, where:
  - The Discriminator is trying to maximize its reward $\boldsymbol{V(D, G)}$
  - The Generator is trying to minimize Discriminator's reward (or maximize its loss)

$$V(D, G) = \boxed{\mathbb{E}_{x\sim p(x)}[\log D(x)]} + \boxed{\mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]}$$

- The Nash equilibrium of this particular game is achieved at:
  - $P_{data}(x) = P_{gen}(x)\ \forall x$
  - $\mathrm{D}(x) = \frac{1}{2}\ \forall x$

Generative Adversarial Networks

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

Discriminator updates

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

Generator updates

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

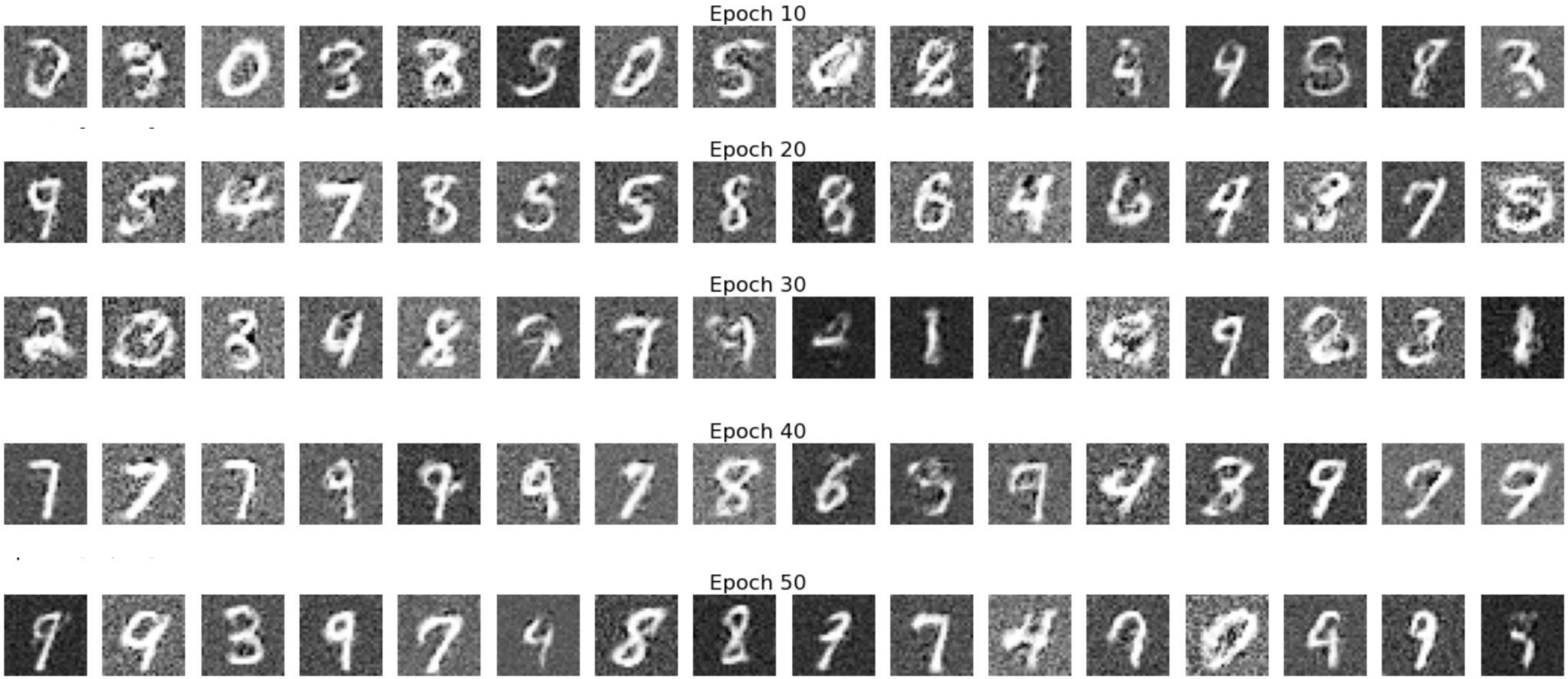
Code Demo

Generative Adversarial Networks

```python
class Generator(nn.Module):
    def __init__(self, z_dim=100):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(z_dim, 128),
            nn.ReLU(True),
            nn.Linear(128, 256),
            nn.ReLU(True),
            nn.Linear(256, 28*28),
            nn.Tanh()  # Output range [-1, 1]
        )

    def forward(self, z):
        out = self.model(z)
        return out.view(z.size(0), 1, 28, 28)

class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 1),
            nn.Sigmoid()  # Probability of real
        )

    def forward(self, img):
        return self.model(img)
```

```python
epochs = 200

for epoch in range(epochs):
    for real_imgs, _ in dataloader:
        real_imgs = real_imgs.to(device)
        batch_size = real_imgs.size(0)

        # === Train Discriminator ===
        D.zero_grad()
        # Real
        real_labels = torch.ones((batch_size, 1), device=device)
        real_output = D(real_imgs)
        d_loss_real = criterion(real_output, real_labels)

        # Fake
        z = torch.randn(batch_size, z_dim, device=device)
        fake_imgs = G(z)
        fake_labels = torch.zeros((batch_size, 1), device=device)
        fake_output = D(fake_imgs.detach())
        d_loss_fake = criterion(fake_output, fake_labels)

        d_loss = d_loss_real + d_loss_fake
        d_loss.backward()
        opt_D.step()

        # === Train Generator ===
        G.zero_grad()
        z = torch.randn(batch_size, z_dim, device=device)
        fake_imgs = G(z)
        real_labels_for_G = torch.ones((batch_size, 1), device=device)  # want D(G(z)) = 1
        g_loss = criterion(D(fake_imgs), real_labels_for_G)
        g_loss.backward()
        opt_G.step()

    print(f"Epoch [{epoch+1}/{epochs}]  D Loss: {d_loss.item():.4f}  G Loss: {g_loss.item():.4f}")

    if (epoch + 1) % 10 == 0:
        show_generated(G, epoch + 1)
```

Generative Adversarial Networks



Problem with GAN: Non-Convergence

- **Deep Learning models (in general) involve a single player**
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - **Problem:** With non-convexity, we might converge to local optima.

$$\min_{G} L(G)$$

- **GANs instead involve two (or more) players**
  - Discriminator is trying to maximize its reward.
  - Generator is trying to minimize Discriminator's reward.

$$\min_{G} \max_{D} V(D, G)$$

  - SGD was not designed to find the Nash equilibrium of a game.
  - **Problem:** We might not converge to the Nash equilibrium at all.

Problem with GAN: Non-Convergence

Non-Convergence

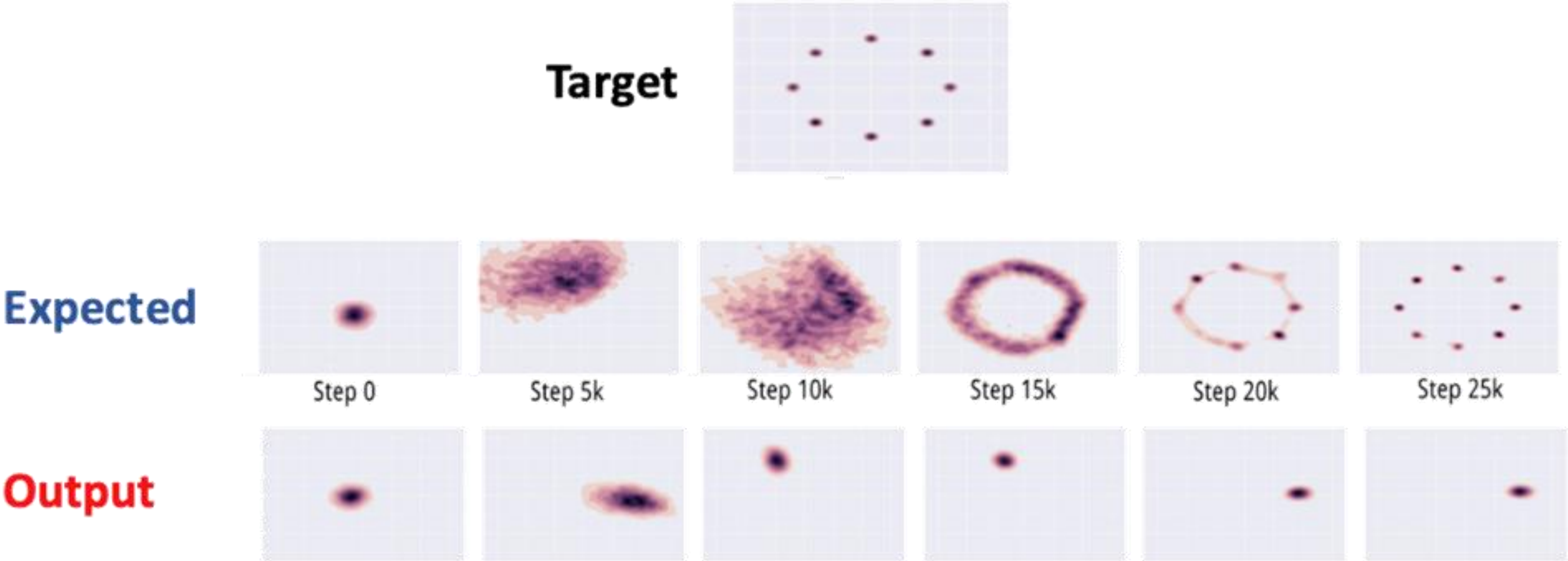
$$\min_{x} \max_{y} V(x, y)$$

Let $V(x, y) = xy$

State 1:	x > 0	y > 0	V > 0	x=2, y=2	Increase y	Decrease x
State 2:	x < 0	y > 0	V < 0	x=-2, y=3	Decrease y	Decrease x
State 3:	x < 0	y < 0	V > 0	x=-3, y=-1	Decrease y	Increase x
State 4 :	x > 0	y < 0	V < 0	x=1, y=-2	Increase y	Increase x
State 5:	x > 0	y > 0	V > 0	== State 1 x=1, y=1	Increase y	Decrease x

Problem with GAN: Mode-Collapse

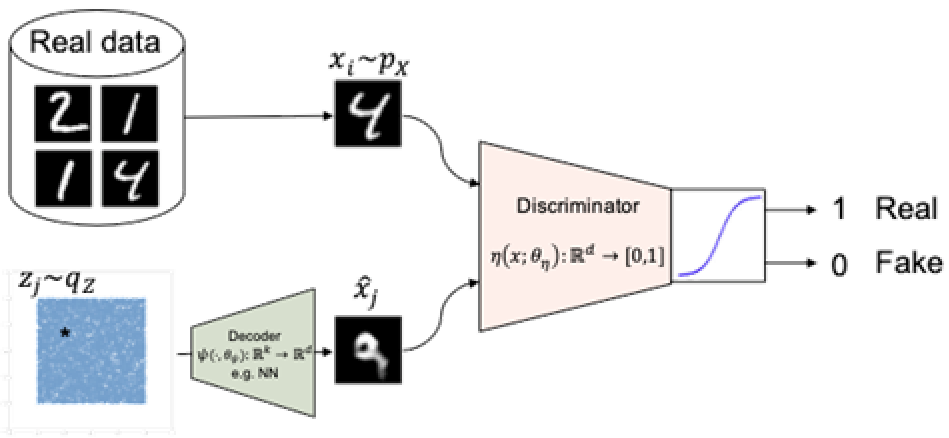
- Generator fails to output diverse samples



Discriminator has achieved the optimal point (can never become more stronger)

Generator therefore will also not be optimized (as discriminator does not give any negative feedback)

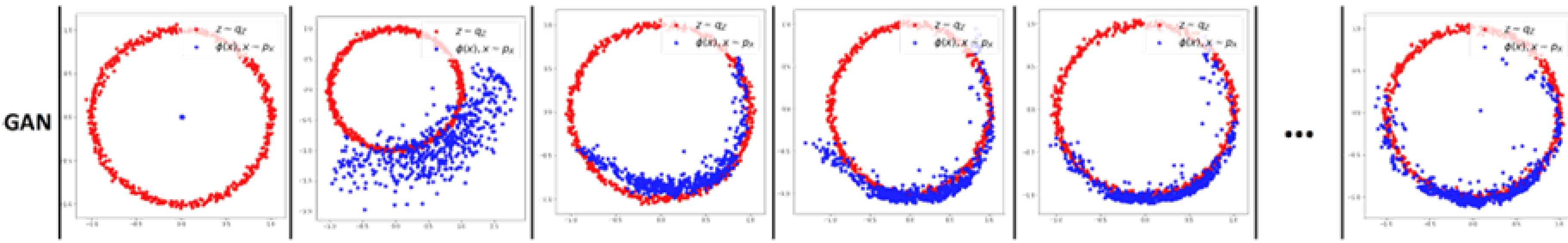
Problem with GAN: Mode-Collapse



GANs have been successfully used in various applications. However, they suffer from two well-known problems:

1. Training GANs could be difficult as there is no guarantee that the backpropagation algorithm achieves the Nash equilibrium.

2. Mode collapse can happen



Problem with GAN: Mode-Collapse

$$\max_{\eta} \mathbb{E}_{p_X}\left[\log\left(\eta(x)\right)\right] + \mathbb{E}_{q_Z}[log(1-\eta(\psi(z)))] = \mathbb{E}_{p_X}\left[\log\left(\eta(x)\right)\right] + \mathbb{E}_{\hat{p}_X}[log(1-\eta(x))]$$

$$\underbrace{\int_X p_X(x)\log\left(\eta(x)\right)dx + \int_X \hat{p}_X(x)\log\left(1-\eta(x)\right)dx}_{L} \Rightarrow \frac{\partial L}{\partial \eta} = \int_X \frac{p_X(x)}{\eta(x)}dx + \int_X \frac{\hat{p}_X(x)}{1-\eta(x)}dx = 0 \Rightarrow \eta^*(x) = \frac{p_X}{p_X+\hat{p}_X}$$

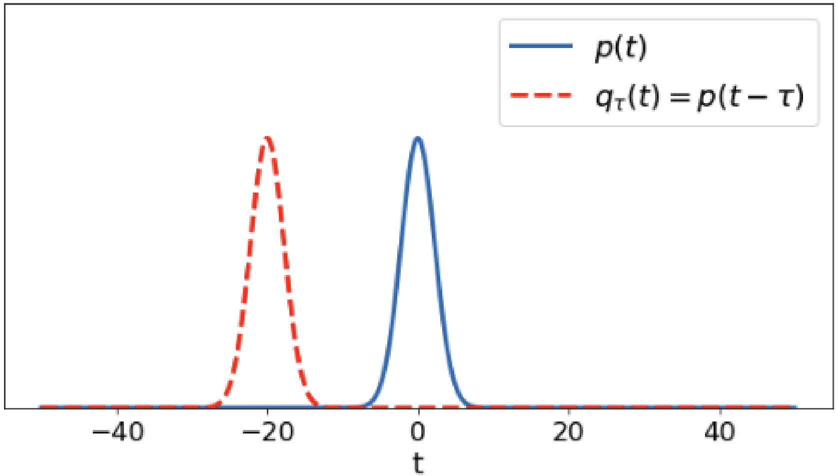
$$\int_X p_X(x)\log\left(\frac{p_X}{p_X+\hat{p}_X}\right)dx + \int_X \hat{p}_X(x)\log\left(1-\frac{p_X}{p_X+\hat{p}_X}\right)dx = \int_X p_X(x)\log\left(\frac{p_X}{p_X+\hat{p}_X}\right)dx + \int_X \hat{p}_X(x)\log\left(\frac{\hat{p}_X}{p_X+\hat{p}_X}\right)dx$$

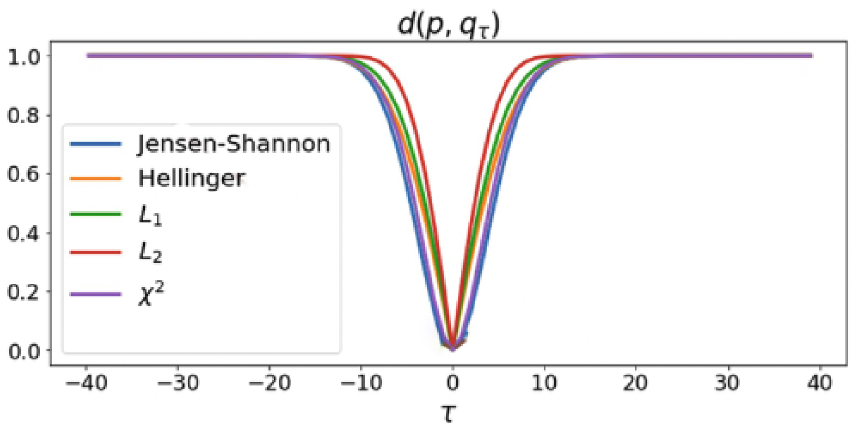
$$= \int_X p_X(x)\log\left(\frac{p_X}{\frac{p_X+\hat{p}_X}{2}}\right)dx + \int_X \hat{p}_X(x)\log\left(\frac{\hat{p}_X}{\frac{p_X+\hat{p}_X}{2}}\right)dx - 2\log(2) = 2JS(p_X,\hat{p}_X) - 2\log(2)$$

For the optimal discriminator, η^*: $\min_{\psi} 2JS(p_X,\hat{p}_X) - 2\log(2)$

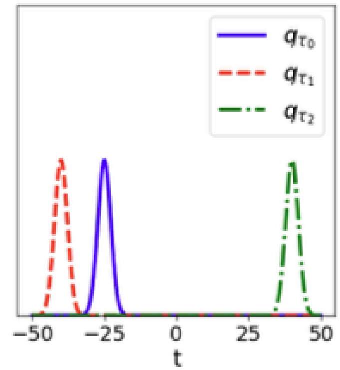
Problem with GAN: Mode-Collapse

Consider the following experiment





Implication 1



$$d(q_{\tau_0}, q_{\tau_1}) = d(q_{\tau_0}, q_{\tau_2}) = d(q_{\tau_1}, q_{\tau_2})$$

The underlying geometry of the space is not captured by these common statistical distances.

Implication 2

Consider the following minimization:

$$argmin_\tau \, d(p, q_\tau)$$

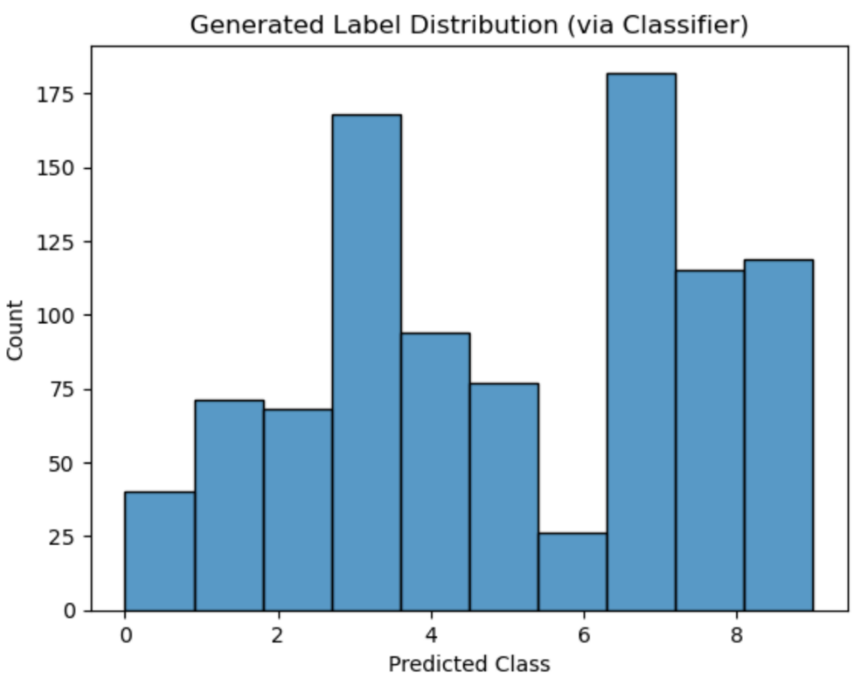
and note that

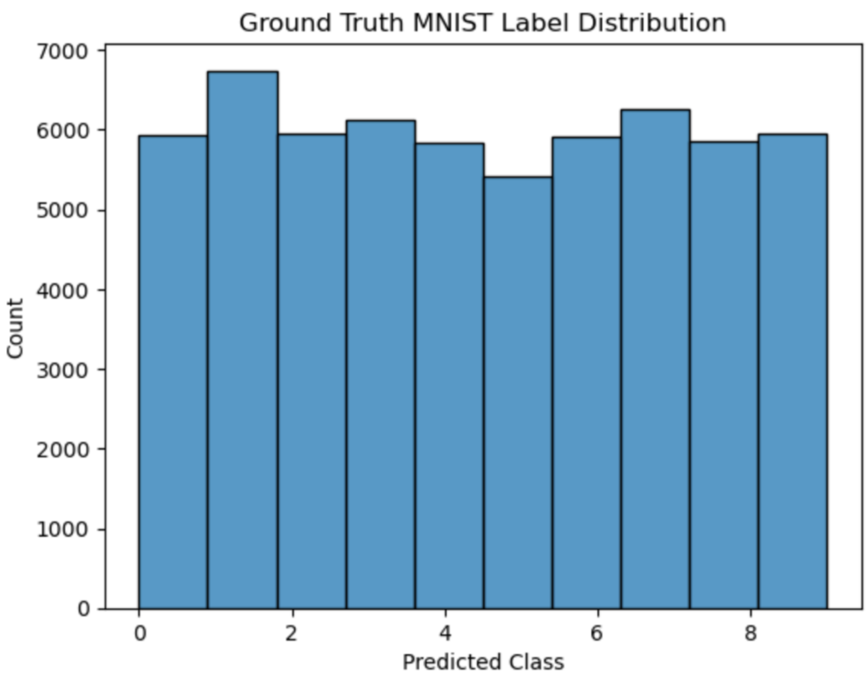
$$\frac{\partial d(p, q_\tau)}{\partial \tau} = 0$$

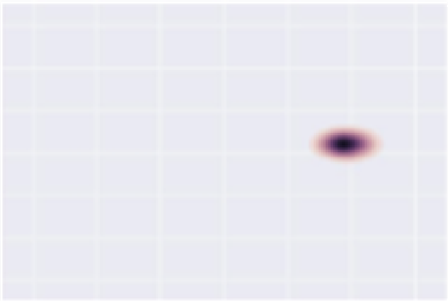
whenever q_τ and p have disjoint domains.

Wasserstein distances capture the underlying geometry of the space and are suitable for learning

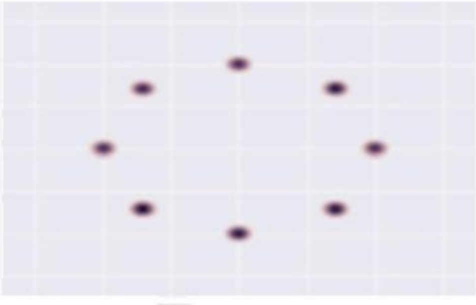
Problem with GAN: Mode-Collapse - Solution







Target



Problem with GAN: Mode-Collapse - Solution

Kantorovich-Rubinstein:

$$W_1(p,\hat{p}) = \max_{\|h\|_L \le 1} E_{x\sim p_X}[h(x)] - E_{x\sim \hat{p}_X}[h(x)]$$

Lipschitz condition: $\|h\|_L \le 1 \rightarrow |h(x_1) - h(x_2)| \le |x_1 - x_2|$

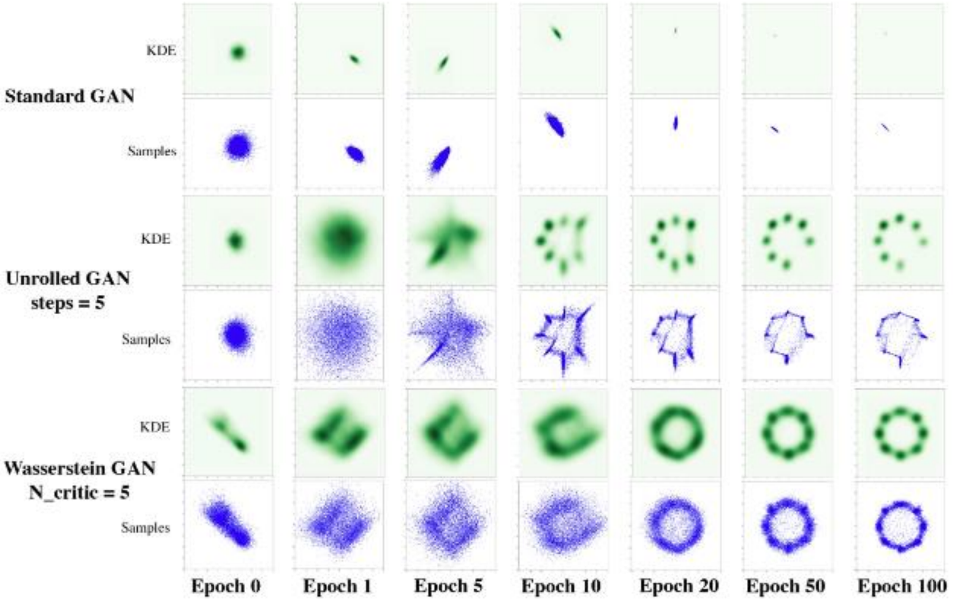
Wasserstein GAN:

$$\min_{\psi} \max_{\|\eta\|_L \le 1} E_{x\sim p_X}[\eta(x)] - E_{x\sim \hat{p}_X}[\eta(x)]$$

Cited 5586 since 2017

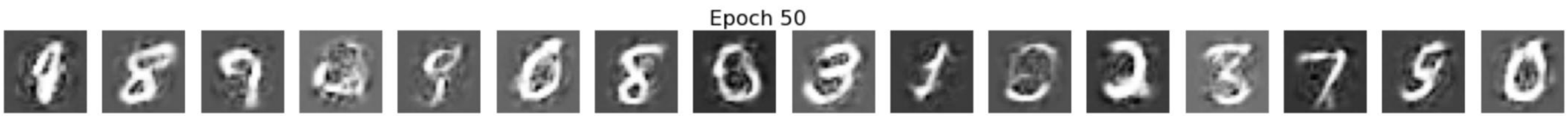
GAN:

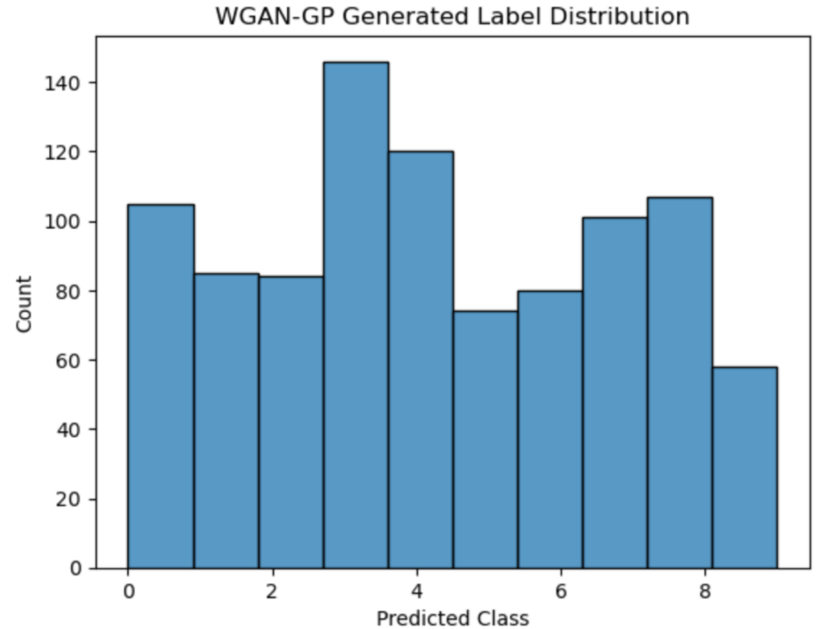
$$\min_{\psi} \max_{\eta} \mathbb{E}_{p_X}[\log(\eta(x))] + \mathbb{E}_{q_Z}[log(1-\eta(\psi(z)))]$$



Problem with GAN: Mode-Collapse - Solution

```
gp = compute_gradient_penalty(C, real_imgs, fake_imgs)
loss_C = -(real_scores.mean() - fake_scores.mean()) + lambda_gp * gp
```





Conditional GANs

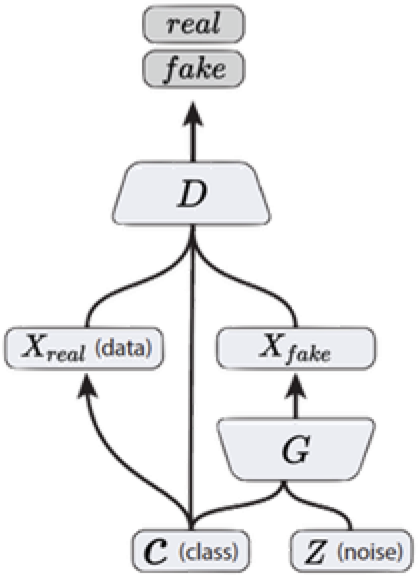
MNIST digits generated conditioned on their class label.

[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] →
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0] →
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0] →
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0] →
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0] →
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0] →
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0] →
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0] →
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0] →
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1] →

Conditional GANs

- Simple modification to the original GAN framework that conditions the model on *additional information* for better multi-modal learning.

- Lends to many practical applications of GANs when we have explicit *supervision* available.



Conditional GAN
(Mirza & Osindero, 2014)

Image Credit: Figure 2 in Odena, A., Olah, C. and Shlens, J., 2016. Conditional image synthesis with auxiliary classifier GANs. *arXiv preprint arXiv:1610.09585*.

Conditional GANs