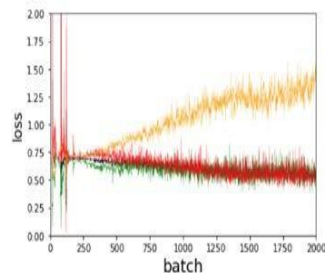


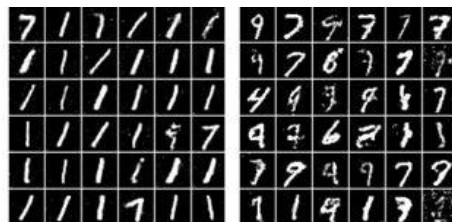
# GAN Training Pathologies

# Summary

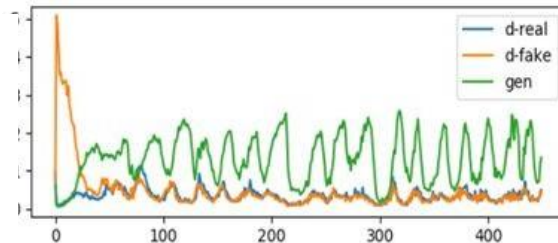
**Diminished gradient:** the discriminator gets too successful that the generator gradient vanishes and learns nothing.



**Mode collapse:** the generator collapses which produces limited varieties of samples.



**Non-convergence:** the model parameters oscillate, destabilize, and never converge.



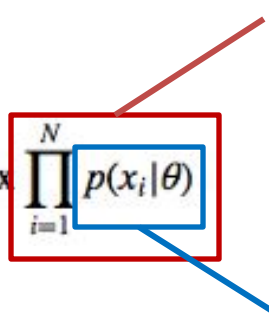
# Summary

Most of this lecture is motivated by:

- **From GAN to WGAN**
  - <https://arxiv.org/abs/1904.08994>
- **Improved Techniques for Training GANs**
  - <https://arxiv.org/abs/1606.03498>

# Generative models training

Many generative models create a model  $\theta$  that maximizes the Maximum Likelihood Estimation **MLE**. i.e. **finding the best model parameters that fit the training data the most.**

$$\hat{\theta} = \arg \max_{\theta} \prod_{i=1}^N p(x_i | \theta)$$


**Likelihood function:** It represents the joint probability of observing the entire dataset  $\{x_1, x_2, \dots, x_N\}$ , assuming the data points are independent.

The probability of a single data point  $x_i$  given the parameters  $\theta$

Find the parameters  $\theta$  that maximize the likelihood of the observed data.

# Generative models training

Many generative models create a model  $\theta$  that maximizes the Maximum Likelihood Estimation **MLE**. i.e. **finding the best model parameters that fit the training data the most.**

$$\hat{\theta} = \arg \max_{\theta} \prod_{i=1}^N p(x_i|\theta)$$

MLE is a statistical method used for estimating the parameters of a model. In the case of generative models, MLE aims to **find the set of parameters ( $\theta$ ) that maximizes the likelihood of observing the training data.**

- Likelihood refers to the probability of observing the given training data under the current model.
- MLE seeks to find the parameters  $\theta$  that make the observed data most probable.

# Generative models training

Maximizing MLE is the same as minimizing the **KL-divergence**  $KL(\mathbf{p}, \mathbf{q})$  which measures how the probability distribution  $\mathbf{q}$  (estimated distribution from the trained model) diverges from the expected probability distribution  $\mathbf{p}$  (the real-life distribution).

Imagine we have two shapes— $\mathbf{p}$  (the real-life distribution) and  $\mathbf{q}$  (estimated distribution). The goal is to adjust the shape of the estimated distribution ( $\mathbf{q}$ ) to match the true shape ( $\mathbf{p}$ ) as closely as possible.

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

**Logarithmic difference** between the probabilities assigned by  $p(x)$  and  $q(x)$  for each possible value of  $x$ .

# Generative models training

Maximizing MLE is the same as minimizing the **KL-divergence**  $KL(\mathbf{p}, \mathbf{q})$  which measures how the probability distribution  $\mathbf{q}$  (estimated distribution from the trained model) diverges from the expected probability distribution  $\mathbf{p}$  (the real-life distribution).

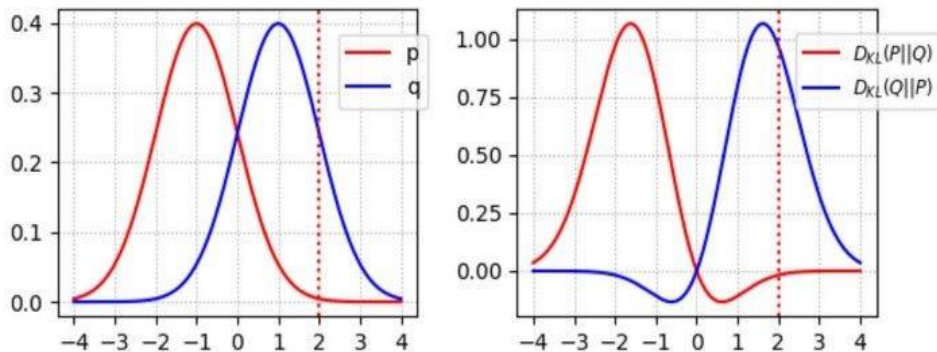
$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

KL-divergence (Kullback-Leibler Divergence) is a mathematical measure that quantifies the difference between two probability distributions.

- $KL(p, q)$ :  $\mathbf{p}$  true probability distribution and  $\mathbf{q}$  is the estimated probability distribution.

The process of minimizing KL-divergence involves adjusting the  $\theta$  parameters of a model to make its estimated distribution,  $\mathbf{q}$ , as close as possible to the true  $\mathbf{p}$ .

# Kullback-Leibler divergence. Main issues



The KL-divergence  $DL(\mathbf{p}, \mathbf{q})$  penalizes the generator if it misses some modes of images:

- The penalty is high where  $p(x) > 0$  but  $q(x) \rightarrow 0$ .

Nevertheless, it is acceptable that some images do not look real. (Poorer quality but more diverse samples)

- The penalty is low when  $p(x) \rightarrow 0$  but  $q(x) > 0$ .

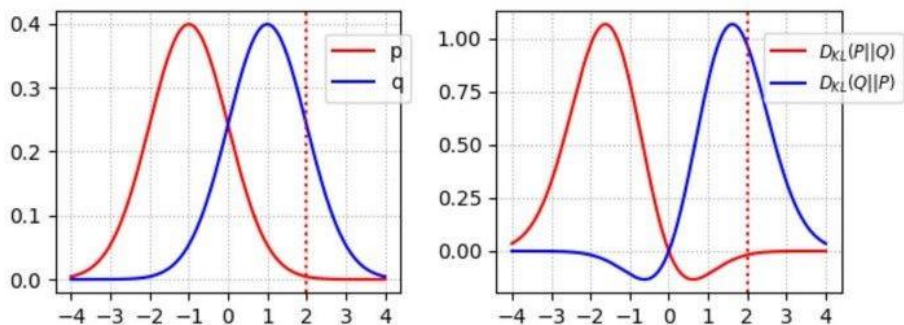
This makes  $DL(\mathbf{p}, \mathbf{q})$  suitable for tasks where ensuring coverage of the real data is critical.

**Issue:** If  $\mathbf{q}$  completely misses  $\mathbf{p}$  in some areas (e.g., **mode collapse**),  $DL(\mathbf{p}, \mathbf{q})$  becomes very large. This can make training unstable or ineffective.



# Kullback-Leibler divergence. Main issues

$KL(x)$  drops to 0 for area where  $p(x) \rightarrow 0$ . For example, in the figure on the right below, the red curve corresponds to  $D(p, q)$ . It drops to zero when  $x > 2$  where  $p$  approaches 0.



The reverse KL-divergence  $DL(q, p)$  penalizes the generator if the images do not look real:

- The penalty is high penalty if  $p(x) \rightarrow 0$  but  $q(x) > 0$ .

But it explores less variety:

- The penalty is low penalty if  $q(x) \rightarrow 0$  but  $p(x) > 0$ . (Better quality but less diverse samples)

**Issue:**  $DL(q, p)$  may encourage the generator to avoid generating “fake” data but might not ensure it covers all modes of  $p$ . This can lead to **mode dropping** (where some parts of the real data distribution are ignored).

# Kullback-Leibler divergence. Main issues

## Mode collapse vs. Mode dropping

- **Mode collapse** occurs when the **generator outputs highly similar (or identical) samples** regardless of the input noise vector  $z$ .
  - The generator ignores the diversity of the noise input  $z$  and only produces samples from a **single mode** (or a small number of modes) in the real data distribution  $p(x)$ .
  - Essentially, the generator “collapses” to one part of the real data distribution and fails to cover other parts.
- **Mode dropping** happens when the **generator captures only a subset of the modes** in the real data distribution but still produces **some diversity**.
  - The generator captures **multiple modes** of the real data distribution but ignores some modes entirely.
  - Unlike mode collapse, there is **still some diversity** in the generator’s output, but it is incomplete.

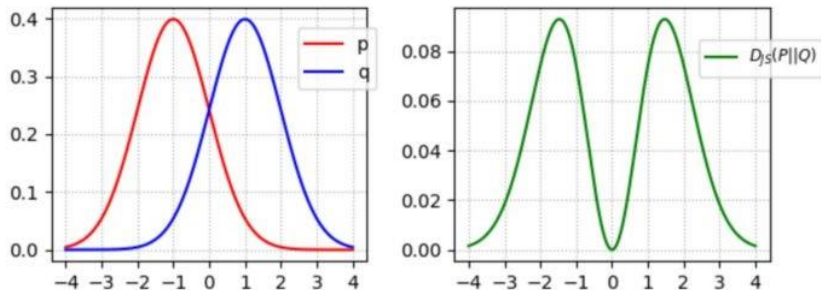
# Jensen-Shannon divergence

Jensen-Shannon divergence is another measure of the difference between two probability distributions, and it's defined in terms of the KL divergence. Specifically, JS divergence is a symmetrized and smoothed version of the KL divergence.

Training GANs has treated as optimizing the generator model is treated as minimizing the JS-divergence.

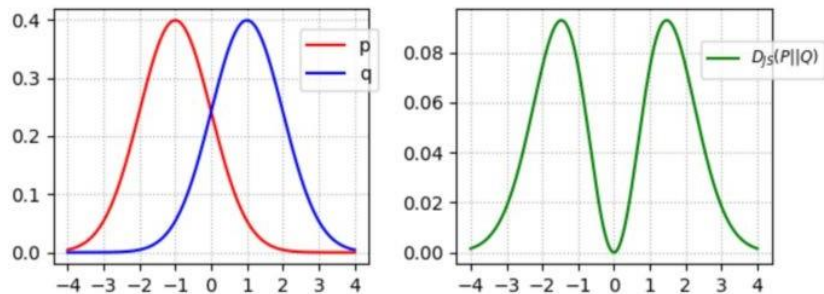
$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2})$$

**Average distribution** (a.k.a., mixture distribution) between p and q



JS divergence combines the KL-divergences of p and q with respect to the average distribution, weighting each by 0.5 (i.e.,  $1/2$ ).

# Jensen-Shannon divergence



$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2})$$

Peaks occur where p and q differ the most.

JS divergence smoothly handles areas where p or q assigns low probability.

## Generator-Discriminator Training:

- The discriminator implicitly estimates the JS divergence between  $p_{\text{data}}$  (real data) and  $p_g$  (generated data).
- As the generator improves,  $p_g$  aligns with  $p_{\text{data}}$ , minimizing the JS divergence.

## Advantages of JS Divergence:

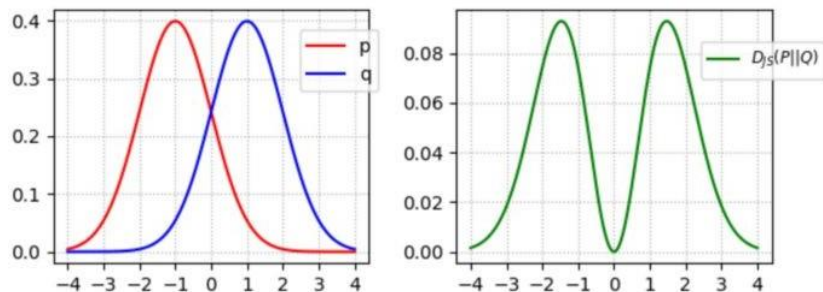
- It gives meaningful gradients even when  $p_g$  and  $p_{\text{data}}$  do not overlap significantly, helping stabilize GAN training.
- Encourages the generator to reduce discrepancies in all regions of the distribution.

# Jensen-Shannon divergence

Jensen-Shannon divergence is another measure of the difference between two probability distributions, and it's defined in terms of the KL divergence. Specifically, JS divergence is a symmetrized and smoothed version of the KL divergence.

Training GANs has treated as optimizing the generator model is treated as minimizing the JS-divergence.

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2})$$



JS-divergence is symmetrical. It will **penalize poor images badly**. (when  $p(x) \rightarrow 0$  and  $q(x) > 0$ )

If the discriminator is optimal (performing well in distinguishing images), the **generator's objective function becomes**

$$\min_G V(D^*, G) = 2D_{JS}(p_r||p_g) - 2 \log 2$$

# Jensen-Shannon divergence

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

$$\begin{aligned} \min_G \max_D L(D, G) &= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_g(z)}[\log(1 - D(G(z)))] \\ &= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))] \end{aligned}$$

$$L(G, D) = \int_x \left( p_r(x) \log(D(x)) + p_g(x) \log(1 - D(x)) \right) dx$$

$$\frac{1}{2} \left( \log 2 + \int_x p_r(x) \log \frac{p_r(x)}{p_r(x) + p_g(x)} dx \right)$$

$$\begin{aligned} D_{JS}(p_r||p_g) &= \frac{1}{2} D_{KL}(p_r||\frac{p_r + p_g}{2}) + \frac{1}{2} D_{KL}(p_g||\frac{p_r + p_g}{2}) \\ &= \frac{1}{2} \left( \log 2 + \int_x p_r(x) \log \frac{p_r(x)}{p_r + p_g(x)} dx \right) + \\ &\quad \frac{1}{2} \left( \log 2 + \int_x p_g(x) \log \frac{p_g(x)}{p_r + p_g(x)} dx \right) \\ &= \frac{1}{2} \left( \log 4 + \underline{L(G, D^*)} \right) \end{aligned}$$

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

The **log 2** comes from normalizing the mixture distribution.

Global optima  $\rightarrow p_r = p_g$

$$\underline{L(G, D^*)} = 2D_{JS}(p_r||p_g) - 2 \log 2$$

$$= -2 \log 2$$

# GAN Training

Generator is trained to minimize JS-Divergence and Discriminator to maximize it. The optimization problem is described as:

$$\begin{aligned}\min_G \max_D L(D, G) &= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \\ &= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]\end{aligned}$$

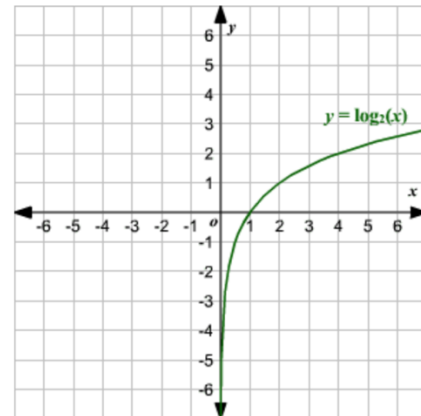
where the first term corresponds to the expectation over real data, and the second term corresponds to the expectation over generated data.

The discriminator's loss is calculated as the sum of the cross-entropy losses for real and generated samples:

$$\mathcal{L}_D^{\text{GAN}} = -\mathbb{E}_{\mathbf{x}} \log D(\mathbf{x}) - \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))$$

The generator's loss is calculated as the negative of the discriminator's loss for generated samples:

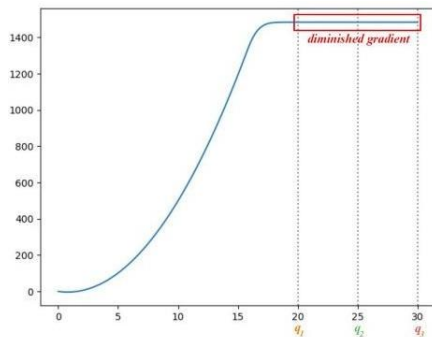
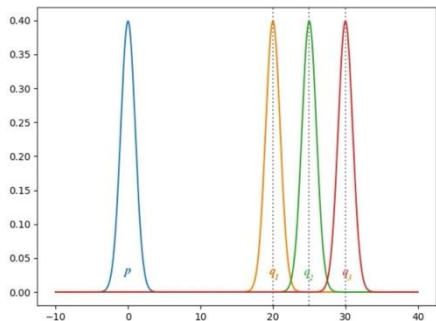
$$\mathcal{L}_G^{\text{GAN}} = \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))$$



# Vanishing gradient

What happens to the JS-divergence gradient when the data distribution  $\mathbf{q}$  of the generator's images does not match with the ground truth  $\mathbf{p}$  for the real images?

Let's consider an example in which  $\mathbf{p}$  and  $\mathbf{q}$  are Gaussian distributed and the mean of  $\mathbf{p}$  is zero. Let's consider  $\mathbf{q}$  with different means to study the gradient of  $JS(p, q)$ .



JS-divergence  $JS(p, q)$   
between  $p$  and  $q$  with  
means of  $q$  ranging from 0  
to 30.

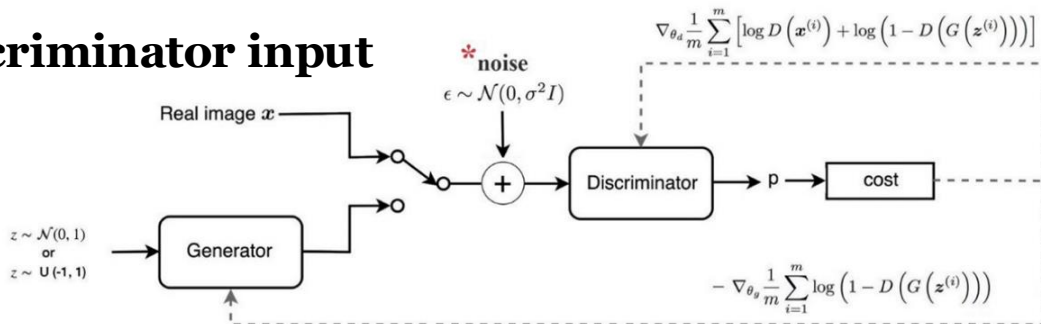
The gradient for the JS-divergence vanishes from  $q_1$  to  $q_3$ . The GAN generator will learn extremely slow to nothing when the cost is saturated in those regions. In particular, in early training,  $p$  and  $q$  are very different and the generator learns very slow.



# Vanishing gradient

Attempts to remedy:

- **Wasserstein loss:** The Wasserstein loss is designed to prevent vanishing gradients even when you train the discriminator to optimality.
- **Modified minimax loss:** The original GAN paper proposed a modification to minimax loss to deal with vanishing gradients.
- **Adding noise to the discriminator input**



# Vanishing gradient

Name	Discriminator Loss	Generator Loss
Minimax GAN	$\mathcal{L}_D^{\text{GAN}} = -\mathbb{E}_{\mathbf{x}} \log D(\mathbf{x}) - \mathbb{E}_{\mathbf{z}} \log (1 - D(G(\mathbf{z})))$	$\mathcal{L}_G^{\text{GAN}} = \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))$
Non-Saturating GAN	$\mathcal{L}_D^{\text{NSGAN}} = \mathcal{L}_D^{\text{GAN}}$	$\mathcal{L}_G^{\text{NSGAN}} = -\mathbb{E}_{\mathbf{z}} \log D(G(\mathbf{z}))$
Least-Squares GAN	$\mathcal{L}_D^{\text{LSGAN}} = \mathbb{E}_{\mathbf{x}} (D(\mathbf{x}) - 1)^2 + \mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))^2$	$\mathcal{L}_G^{\text{LSGAN}} = \mathbb{E}_{\mathbf{z}} (D(G(\mathbf{z})) - 1)^2$
Wasserstein GAN	$\mathcal{L}_D^{\text{WGAN}} = -\mathbb{E}_{\mathbf{x}} D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))$	$\mathcal{L}_G^{\text{WGAN}} = -\mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))$
WGAN-GP	$\mathcal{L}_D^{\text{WGANGP}} = \mathcal{L}_D^{\text{WGAN}} + \lambda \mathbb{E}_{\mathbf{x}, \mathbf{z}} (\ \nabla D(\alpha \mathbf{x} + (1 - \alpha)G(\mathbf{z}))\ _2 - 1)^2$	$\mathcal{L}_G^{\text{WGANGP}} = \mathcal{L}_G^{\text{WGAN}}$
DRAGAN	$\mathcal{L}_D^{\text{DRAGAN}} = \mathcal{L}_D^{\text{GAN}} + \lambda \mathbb{E}_{\mathbf{x} \sim p_{\text{data}} + \mathcal{N}(0, c)} (\ \nabla D(\mathbf{x})\ _2 - 1)^2$	$\mathcal{L}_G^{\text{DRAGAN}} = \mathcal{L}_G^{\text{GAN}}$
BEGAN	$\mathcal{L}_D^{\text{BEGAN}} = \mathbb{E}_{\mathbf{x}} \ \mathbf{x} - \text{AE}(\mathbf{x})\ _1 - k_t \mathbb{E}_{\mathbf{z}} \ G(\mathbf{z}) - \text{AE}(G(\mathbf{z}))\ _1$	$\mathcal{L}_G^{\text{BEGAN}} = \mathbb{E}_{\mathbf{z}} \ G(\mathbf{z}) - \text{AE}(G(\mathbf{z}))\ _1$

# Mode collapse

Mode collapse happens when the generator in a GAN learns to produce only a **limited range of outputs** or, in the extreme case, a **single output, no matter the input noise  $\mathbf{z}$** . This happens because the generator focuses on “fooling” the discriminator by creating one specific image or set of images that the discriminator finds realistic, ignoring the rest of the real data distribution.

The generator’s goal is to **maximize its chances of fooling the discriminator**. This means:

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

The generator updates its parameters ( $\theta_g$ ) to make the discriminator **D** think its generated outputs (**G(z)**) are real.

# Mode collapse

**Extreme case:**  $G$  is trained extensively **without updates to  $D$** . The generated images will converge to find the optimal image  $\mathbf{x}^*$  that deceives  $D$  the most, the most realistic image from the discriminator perspective. **In this extreme,  $\mathbf{x}^*$  will be independent of  $\mathbf{z}$ .**

$$x^* = \operatorname{argmax}_x D(x)$$

The generator essentially ignores the noise  $\mathbf{z}$  because it has found an “optimal” solution to deceive the discriminator with one specific output.

# Mode collapse

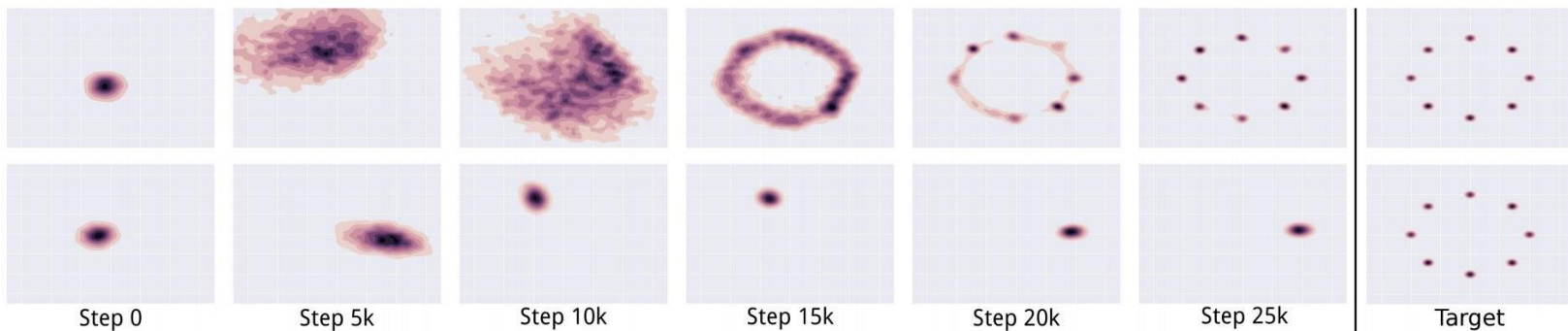
This is bad news. The mode collapses to a **single point**. The gradient associated with  $\mathbf{z}$  approaches zero.

$$\frac{\partial J}{\partial \mathbf{z}} \approx 0$$

- Restart the training in the discriminator: **the most effective way to detect generated images is to detect this single mode.**
- Since the generator has minimized the impact of  $\mathbf{z}$  already in the generated samples, **the gradient from the discriminator will likely push the single point around for the next most vulnerable mode.**
  - This is not hard to find. The generator produces such an imbalance of modes in training that it deteriorates its capability to detect others.
- Now, **both networks are overfitted to exploit short-term opponent weakness.**

# Mode collapse

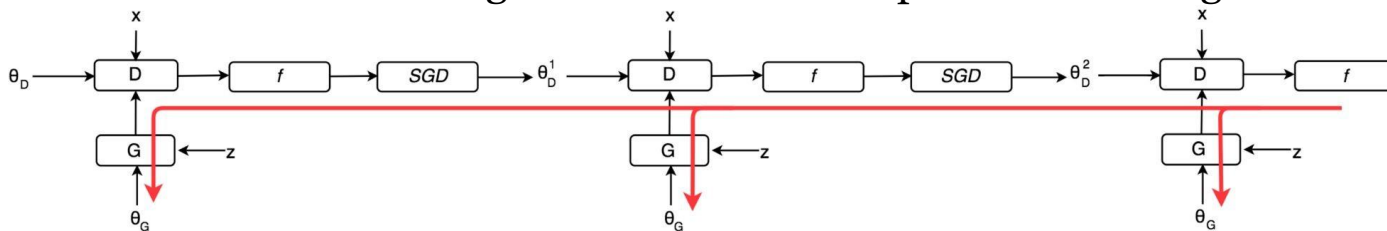
- Restart the training in the discriminator: **the most effective way to detect generated images is to detect this single mode.**
- Since the generator has minimized the impact of  $\mathbf{z}$  already in the generated samples, **the gradient from the discriminator will likely push the single point around for the next most vulnerable mode.**
  - This is not hard to find. The generator produces such an imbalance of modes in training that it deteriorates its capability to detect others.



# Mode collapse

Attempts to remedy:

- **Wasserstein loss:** The Wasserstein loss alleviates mode collapse by letting you train the discriminator to optimality without worrying about vanishing gradients. If the discriminator doesn't get stuck in local minima, it learns to reject the outputs that the generator stabilizes on. So the generator has to try something new.
- **Unrolled GANs:** Unrolled GANs use a generator loss function that incorporates not only the current discriminator's classifications, but also the outputs of future discriminator versions. So the generator can't over-optimize for a single discriminator.



# Mode collapse

Attempts to remedy:

- **Regularization Techniques:** Techniques such as dropout or weight regularization can be employed to prevent the model from becoming too specialized in certain patterns.
- **Balancing Training Dynamics:** Carefully tuning the learning rates of the generator and discriminator or employing techniques like **curriculum learning** can help balance the adversarial training dynamics.
  - Curriculum learning is a training strategy where the learning algorithm is exposed to a curriculum, or a sequence of samples, in a specific order of increasing complexity or difficulty.



# Oscillation

In GAN training, oscillation occurs when the generator and discriminator are constantly **chasing each other** without ever reaching a stable point (equilibrium). Instead of improving, their outputs keep swinging back and forth, leading to instability.

GAN is based on the **zero-sum non-cooperative game** (if one wins the other loses) also called minimax. Your opponent wants to maximize its actions and your actions are to minimize them.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_r(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

In game theory, the **GAN model converges** when the discriminator and the generator reach a **Nash equilibrium**. Nash equilibrium happens when one player will not change its action regardless of what the opponent may do.

# Oscillation

Consider two player  $A$  and  $B$  which control the value of  $x$  and  $y$ , respectively. Player  $A$  wants to maximize the value  $xy$  while  $B$  wants to minimize it.

$$\min_B \max_A V(D, G) = xy$$

The Nash equilibrium is  $x=y=0$ .

# Oscillation

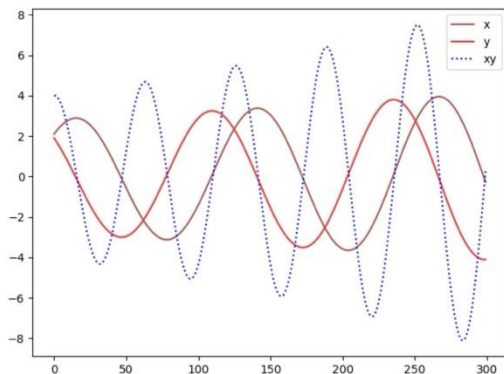
Let's see whether we can find the Nash equilibrium easily using the gradient descent. We update the parameter  $\mathbf{x}$  and  $\mathbf{y}$  based on the gradient of the value function  $V$ ,  $V(x,y) = xy$

$$\Delta x = \alpha \frac{\partial(xy)}{\partial(x)}$$

$$\Delta y = -\alpha \frac{\partial(xy)}{\partial(y)}$$

where  $\alpha$  is the learning rate

When we plot  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{xy}$  against the training iterations, **we realize our solution does not converge.**



It is an excellent showcase that **some cost functions will not converge with gradient descent**, in particular for a non-convex game.

# Oscillation

- GAN training is like a **non-convex optimization problem** (like a bumpy hill), where the generator and discriminator are playing a “game” instead of working towards a single shared goal.
- Gradient descent works poorly in these settings, as it doesn’t guide the system to a stable solution.

## Attempts to remedy:

Researchers have tried to use various forms of **regularization** to improve GAN convergence, including:

- **Adding noise to discriminator inputs**
- **Penalizing discriminator weights**

# Improving GAN training

---

## Improved Techniques for Training GANs

---

**Tim Salimans**  
tim@openai.com

**Ian Goodfellow**  
ian@openai.com

**Wojciech Zaremba**  
woj@openai.com

**Vicki Cheung**  
vicki@openai.com

**Alec Radford**  
alec@openai.com

**Xi Chen**  
peter@openai.com

**Abstract**

<https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b>

<https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html>