

Lecture 15:

Generative Models - GAN

Deep Learning

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

Lecture Contents

Part 1: Introduction and Motivation

Part 2: Mathematical Foundations

Part 3: Training Algorithm

Part 4: Key Challenges

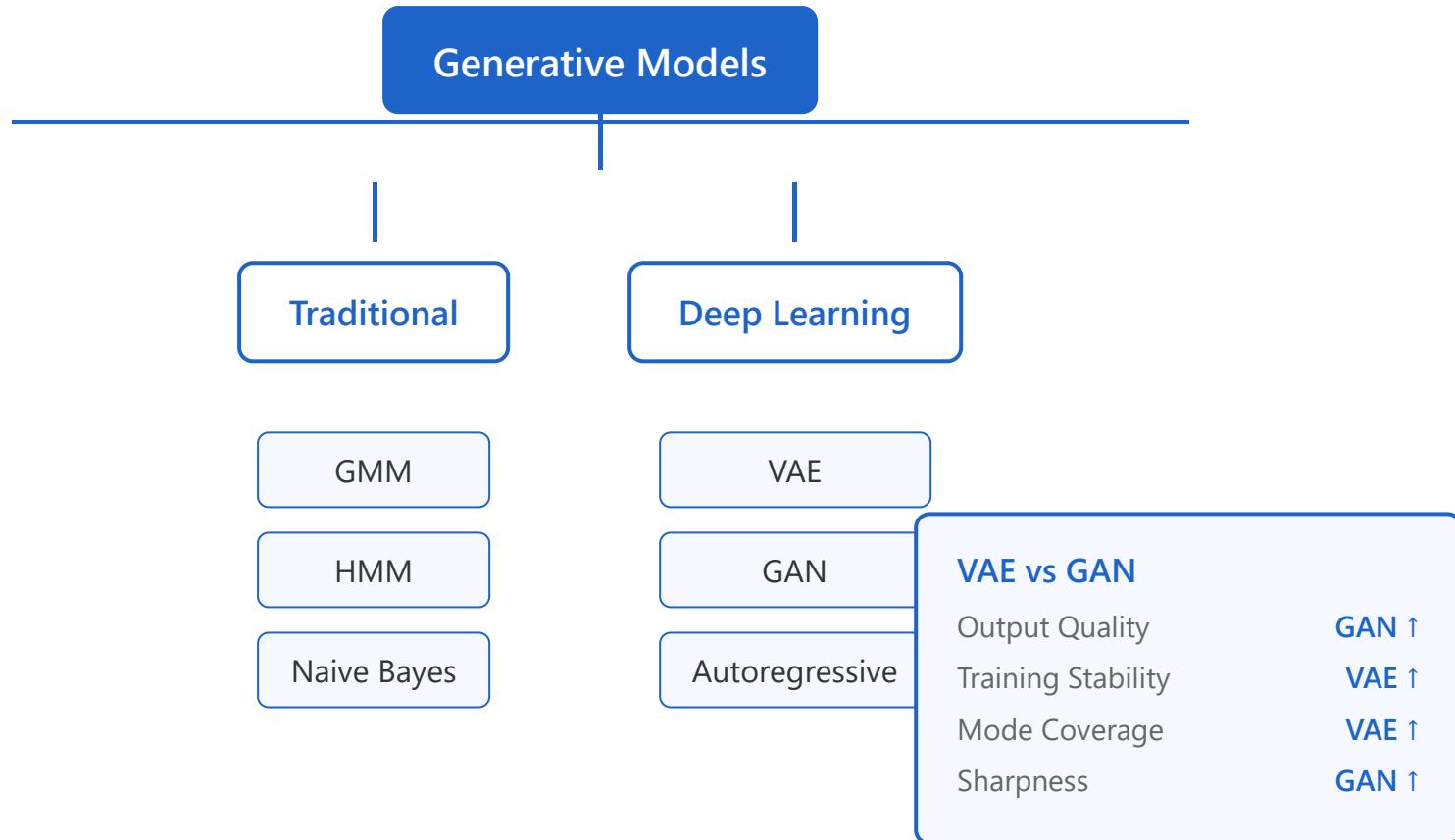
Part 5: Improvement Techniques

Part 1/6:

Introduction & Motivation

1. What are Generative Models?
2. Why GANs?
3. GAN Architecture Overview

Generative Models Taxonomy



1. GMM (Gaussian Mixture Model)

📌 Basic Principle

GMM is a probabilistic model that assumes data is generated from a **mixture of multiple Gaussian distributions (normal distributions)**. Each Gaussian component represents a specific cluster in the data, and the model calculates the probability that each data point was generated from which component.

$$p(x) = \sum \pi_k \cdot N(x | \mu_k, \Sigma_k)$$

Here, π_k is the mixing coefficient of the k-th component, μ_k is the mean, and Σ_k is the covariance matrix. Parameters are learned using the EM (Expectation-Maximization) algorithm.

Main Applications

- **Clustering:** Modeling more flexible cluster shapes than K-means
- **Image Segmentation:** Segmenting images into multiple regions
- **Anomaly Detection:** Detecting data that deviates from normal data distribution
- **Speech Recognition:** Phoneme modeling
- **Density Estimation:** Approximating complex data distributions

 **Advantages:** Soft clustering (probabilistic assignment), capable of modeling elliptical clusters

 **Disadvantages:** Number of components must be specified in advance, performance degrades in high-dimensional data

2. HMM (Hidden Markov Model)

Basic Principle

HMM is a probabilistic model for **time series data**, assuming that observable events are generated by hidden states. The system transitions between invisible states and generates observations from each state.

Core Components:

- **Transition Probability (A):** Probability of moving from one state to another
- **Emission Probability (B):** Probability of an observation occurring in a specific state
- **Initial State Probability (π):** Probability of each state at the start

$$P(O|\lambda) = \sum P(O|Q, \lambda) \cdot P(Q|\lambda)$$

Uses **Forward-Backward algorithm** (inference), **Viterbi algorithm** (optimal path search), and **Baum-Welch algorithm** (learning).

Main Applications

- **Speech Recognition:** Recognizing words from phoneme sequences
- **Natural Language Processing:** Part-of-speech (POS) tagging
- **Bioinformatics:** Gene sequence analysis, DNA pattern recognition
- **Gesture Recognition:** Recognizing motion patterns in video
- **Financial Time Series:** Stock pattern and market state analysis

 **Advantages:** Models temporal dependencies, can handle incomplete observation data

 **Disadvantages:** Markov assumption (present depends only on previous state), difficulty in choosing number of states

3. Naive Bayes

Basic Principle

Based on Bayes' theorem, it makes the "naive" assumption that all features are **conditionally independent**. While this assumption is unrealistic, it works surprisingly well in practice.

$$P(y|x_1, \dots, x_n) = P(y) \cdot \prod P(x_i|y) / P(x_1, \dots, x_n)$$

Given class y , it assumes each feature x_i is independent. This allows decomposing high-dimensional joint probabilities into simple products of conditional probabilities, making computation very efficient.

Main Variants:

- **Gaussian NB:** For continuous data, features follow normal distribution
- **Multinomial NB:** For text classification, based on word frequency
- **Bernoulli NB:** For binary features, word presence/absence

Main Applications

- **Text Classification:** Spam filtering, sentiment analysis, news categorization
- **Document Classification:** Email classification, document categorization

- **Recommendation Systems:** Predicting user preferences
- **Medical Diagnosis:** Calculating disease probability based on symptoms
- **Real-time Prediction:** Classification tasks requiring fast speed

 **Advantages:** Very fast and efficient, can learn with small data, easy to interpret

 **Disadvantages:** Feature independence assumption is unrealistic, ignores correlations between features

4. VAE (Variational Autoencoder)

Basic Principle

VAE is a deep learning generative model that combines an **autoencoder architecture** with **variational inference**. It encodes data into a low-dimensional latent space and reconstructs data from this latent representation.

Core Architecture:

- **Encoder ($q_\varphi(z|x)$):** Maps input data x to probability distribution of latent variable z (mean μ , variance σ^2)
- **Reparameterization Trick:** Samples $z = \mu + \sigma \odot \varepsilon$ ($\varepsilon \sim N(0,1)$) to enable backpropagation
- **Decoder ($p_\theta(x|z)$):** Reconstructs original data from latent variable z

Loss Function (ELBO)

VAE is trained by maximizing the Evidence Lower Bound (ELBO), which balances two terms:

$$L = E[\log p_\theta(x|z)] - KL(q_\varphi(z|x) || p(z))$$

First term (Reconstruction Loss): How well the decoder reconstructs the original data

Second term (KL Divergence): Difference between the learned distribution and prior distribution (usually $N(0,1)$)



Key Insight: The KL term regularizes the latent space to make it smooth and continuous, enabling meaningful interpolation when generating new samples.

Properties of Latent Space

VAE's latent space has the following important properties:

- **Continuity:** Similar data points are located close in latent space
- **Interpolability:** Smooth movement between two points generates meaningful intermediate samples
- **Structured:** Specific dimensions control specific attributes (e.g., facial expression, angle)
- **Regularity:** Regularization to prior distribution reduces empty spaces

Main Applications

- **Image Generation:** Generating new faces, landscapes, artworks
- **Data Augmentation:** Generating transformed images to expand training data
- **Anomaly Detection:** Identifying samples with high reconstruction error as anomalies
- **Dimensionality Reduction:** Capturing non-linear relationships better than t-SNE, PCA
- **Image Editing:** Changing specific attributes by manipulating latent vectors
- **Semi-supervised Learning:** Learning representations with limited labels
- **Drug Design:** Exploring latent space of molecular structures



Detailed VAE vs GAN Comparison

VAE Strengths:

- **Stable Training:** Single objective function, no mode collapse
- **Complete Probabilistic Model:** Explicit probability density estimation
- **Interpretable Latent Space:** Learning structured and meaningful representations
- **Mode Coverage:** Better capturing diverse data distributions

VAE Weaknesses:

- **Blurry Output:** Reconstruction loss (MSE) generates averaged images
- **Lower Sample Quality:** Generated images are less sharp than GAN's
- **Prior Distribution Assumption:** Usually assumes Gaussian, which may differ from actual distribution



Practical Tip: Use β -VAE (weight β on KL term) to adjust balance between reconstruction and regularization. $\beta > 1$ learns more disentangled representations but may reduce reconstruction quality.

💡 Main Variants

- **β -VAE:** KL weight adjustment for disentangled representation learning
- **Conditional VAE (CVAE):** Adding label information for conditional generation
- **VQ-VAE:** Using discrete latent space, high-quality image/audio generation
- **Hierarchical VAE:** Modeling complex structures with multi-level latent variables



Input/Output Calculation Example

Let's examine a VAE that encodes MNIST handwritten digit images (28×28) into a 2-dimensional latent space.

Network Structure:

- **Input:** $28 \times 28 = 784$ -dimensional image vector
- **Encoder:** $784 \rightarrow 400 \rightarrow (\mu: 2\text{-dim}, \log \sigma^2: 2\text{-dim})$
- **Latent space:** 2-dimensional (good for visualization)
- **Decoder:** $2 \rightarrow 400 \rightarrow 784$
- **Output:** 28×28 reconstructed image

⟳ Forward Pass Calculation Process

Step 1: Input Image

$$x \in \mathbb{R}^{(784)} = [0.1, 0.9, 0.8, \dots, 0.0] \text{ (normalized pixel values)}$$

Step 2: Encoder - Hidden Layer

$$h = \text{ReLU}(W_1 \cdot x + b_1) \in \mathbb{R}^{(400)}$$

Example: Using $W_1 \in \mathbb{R}^{(400 \times 784)}$, $b_1 \in \mathbb{R}^{(400)}$ to compress 784 dimensions to 400 dimensions

Step 3: Encoder - Computing Mean and Variance

$$\begin{aligned}\mu &= W_{\mu} \cdot h + b_{\mu} \in \mathbb{R}^2 = [1.2, -0.8] \\ \log \sigma^2 &= W_{\sigma} \cdot h + b_{\sigma} \in \mathbb{R}^2 = [-0.5, -1.2]\end{aligned}$$

We predict $\log \sigma^2$ for numerical stability.

Actual standard deviation: $\sigma = \exp(0.5 \cdot \log \sigma^2) = [0.78, 0.55]$

Step 4: Reparameterization Trick

$$\begin{aligned}\varepsilon &\sim N(0, I) \in \mathbb{R}^2 = [0.3, -0.7] \text{ (random sampling)} \\ z &= \mu + \sigma \odot \varepsilon = [1.2, -0.8] + [0.78, 0.55] \odot [0.3, -0.7] \\ z &= [1.2 + 0.234, -0.8 - 0.385] = [1.434, -1.185]\end{aligned}$$

 **Key Point:** By sampling ε , backpropagation becomes possible. We can calculate gradients with respect to μ and σ .

Step 5: Decoder - Hidden Layer

$$h' = \text{ReLU}(W_2 \cdot z + b_2) \in \mathbb{R}^{(400)}$$

Example: Using $W_2 \in \mathbb{R}^{(400 \times 2)}$, $b_2 \in \mathbb{R}^{(400)}$ to expand 2 dimensions to 400 dimensions

Step 6: Decoder - Output (Reconstruction)

$$\hat{x} = \sigma(W_3 \cdot h' + b_3) \in \mathbb{R}^{(784)}$$

σ is the sigmoid function that constrains output to $[0, 1]$ range.

$\hat{x} = [0.09, 0.88, 0.82, \dots, 0.01]$ (reconstructed image)

Loss Calculation

Reconstruction Loss:

$$L_{\text{recon}} = -\sum_i [x_i \cdot \log(\hat{x}_i) + (1-x_i) \cdot \log(1-\hat{x}_i)] \text{ (Binary Cross-Entropy)}$$

Or MSE can be used:

$$L_{\text{recon}} = \sum_i (x_i - \hat{x}_i)^2 / 784$$

Example: $L_{\text{recon}} = 0.025$ (average error per pixel)

KL Divergence:

$$KL = -0.5 \cdot \sum_j [1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2]$$

Calculating per dimension:

$$\begin{aligned} j=1: -0.5 \cdot [1 + (-0.5) - 1.2^2 - e^{-(-0.5)}] &= -0.5 \cdot [0.5 - 1.44 - 0.61] = 0.775 \\ j=2: -0.5 \cdot [1 + (-1.2) - 0.8^2 - e^{-(-1.2)}] &= -0.5 \cdot [-0.2 - 0.64 - 0.30] = 0.570 \\ KL_{\text{total}} &= 0.775 + 0.570 = 1.345 \end{aligned}$$

Total Loss:

$$L_{\text{total}} = L_{\text{recon}} + KL = 0.025 + 1.345 = 1.370$$

💡 Interpretation:

- Low reconstruction loss (0.025) → Good reconstruction of image
- High KL (1.345) → Learned distribution deviates significantly from standard normal
- As training progresses, both losses will balance out

🎨 Generating New Images

After training is complete, to generate new images:

1. Sample from standard normal: $z_{\text{new}} \sim N(0, I) = [-0.5, 1.2]$
2. Use decoder only: $x_{\text{new}} = \text{Decoder}(z_{\text{new}})$
3. Generate new handwritten digit image!

Interpolation in latent space is also possible:

$$z_{\text{interpolate}} = \alpha \cdot z_1 + (1-\alpha) \cdot z_2, \alpha \in [0, 1]$$

Example: When $\alpha=0.5$, $z = 0.5 \cdot [1.4, -1.2] + 0.5 \cdot [-0.8, 0.9] = [0.3, -0.15]$

Practical Examples:

- Generate from $z = [2.0, 0.0]$ → Image of '1' with thick lines
- Generate from $z = [0.0, 2.0]$ → Round '0' image
- Generate from $z = [1.0, 1.0]$ → Intermediate form mixing both styles

💡 Parameter Count Calculation:

- Encoder: $784 \times 400 + 400 + 400 \times 2 + 2 + 400 \times 2 + 2 = 315,604$
- Decoder: $2 \times 400 + 400 + 400 \times 784 + 784 = 315,184$
- Total parameters: ~630K (relatively small model)

Birth of GANs



2013

VAE Era



2014

GAN Born



2015+

Rapid Growth

Original Paper

"Generative Adversarial Networks"

Authors: Ian Goodfellow et al.

Published: NIPS 2014

Institution: Université de Montréal

Citations: 50,000+ (Most cited AI paper)

Key Innovation

First framework to train generative models through **adversarial process** - two neural networks competing in a game-theoretic scenario. No need for explicit density estimation or Markov chains.

Revolutionary Impact on AI

10+

Years of Innovation

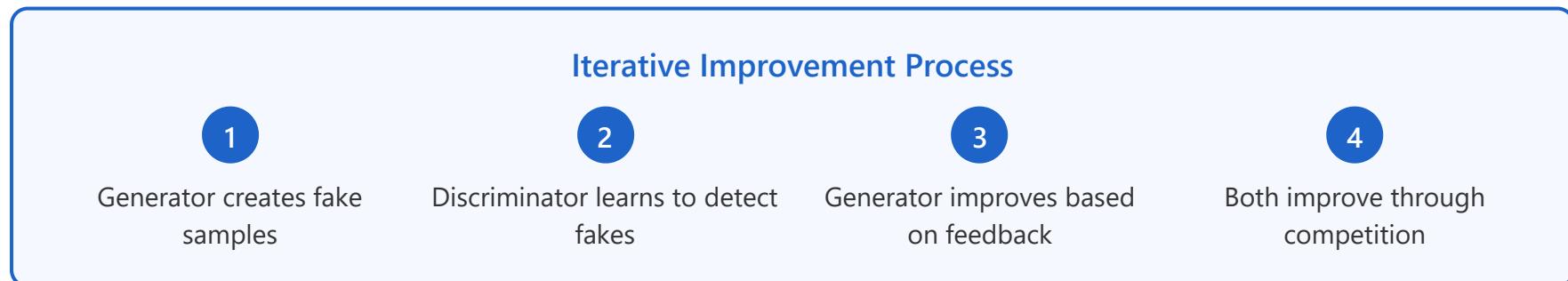
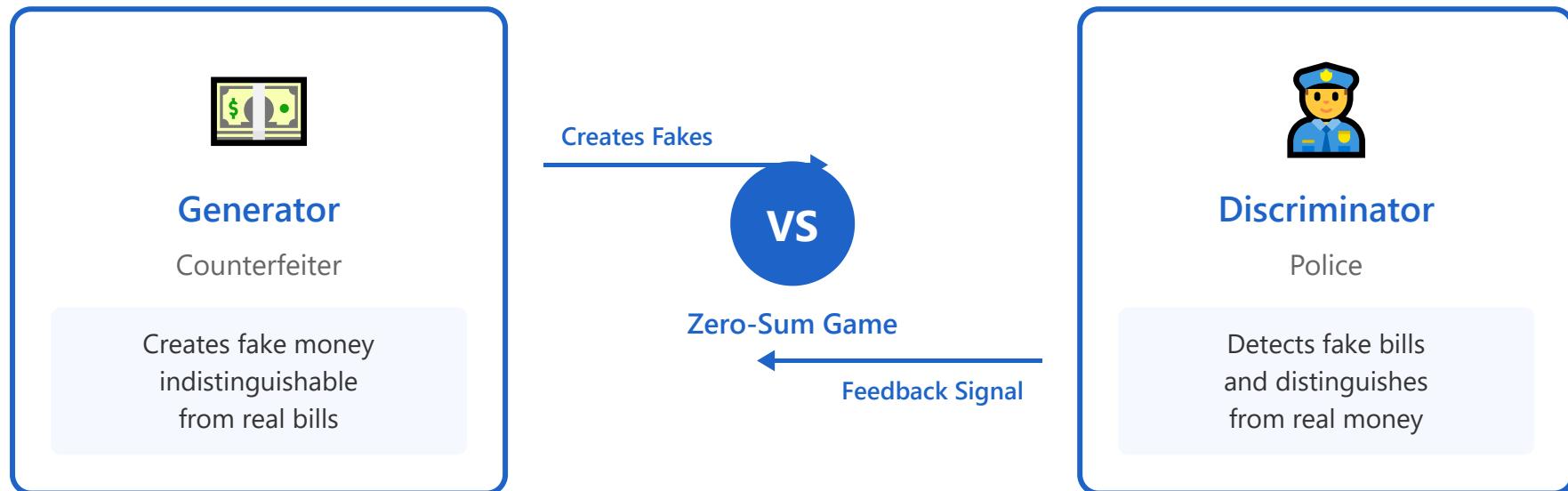
1000+

GAN Variants

∞

Applications

GAN: The Counterfeiter Analogy



Equilibrium

Perfect fakes become
indistinguishable from real data

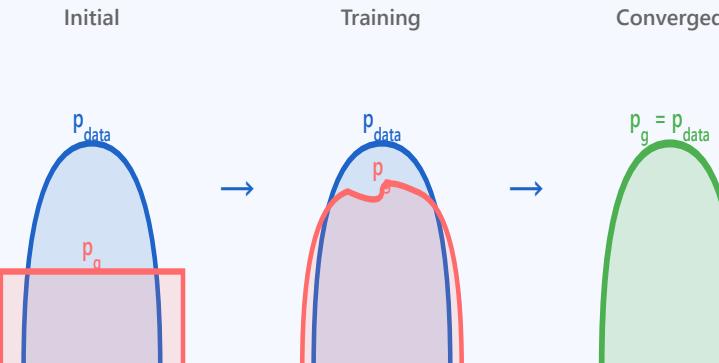
Part 2/6:

Mathematical Foundations

- 4.** Probability Distribution Perspective
- 5.** Mathematical Definition of GAN
- 6.** Value Function Analysis
- 7.** Deriving Optimal Discriminator
- 8.** Global Optimum

Probability Distribution Perspective

Distribution Convergence Process



Real Data Distribution

$p_{\text{data}}(x)$ - True distribution of training data

Generator Distribution

$p_g(x)$ - Distribution learned by generator

Objective

Minimize distance between distributions

$$D(p_{\text{data}} \parallel p_g) \rightarrow 0$$

Sampling Process

- z Sample from latent space: $z \sim p_z(z)$
- G Neural network transformation: $G(z)$
- x Generate sample: $x \sim p_g(x)$

Key Advantages

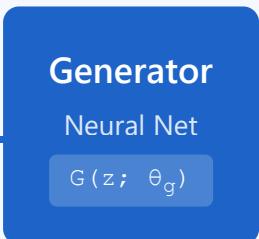
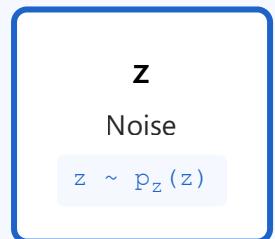
- ✓ Implicit density modeling
- ✓ No explicit likelihood computation
- ✓ Direct sampling capability

Mathematical Definition of GAN

Minimax Objective

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

Two-Player Game Architecture



Generator G

Maps noise to data space

Discriminator D

Outputs probability [0,1]

Training

Alternating optimization

Solution

Nash equilibrium

Value Function $V(D, G)$

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

Real Data Term

D classifies real as real

Fake Data Term

D classifies fake as fake

Value Function Analysis



Discriminator Maximizes

Correctly classify real and fake samples

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



Generator Minimizes

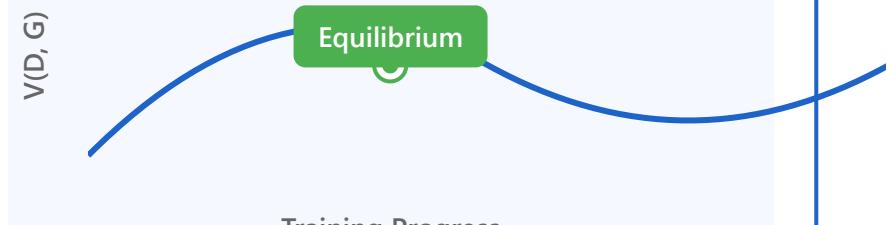
Fool the discriminator effectively

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

Theoretical Insights

- Cross-entropy loss interpretation
- Jensen-Shannon divergence connection
- Global optimality guarantees

Value Function Landscape



Optimal Discriminator

$$D^*(x) = p_{\text{data}}(x) / (p_{\text{data}}(x) + p_g(x))$$

Theoretical Optimum

Discriminator at Equilibrium

Value Function

$$p_g = p_{\text{data}}$$

$$D^*(x) = 1/2$$

$$V(D^*, G^*) = -\log(4)$$

Deriving Optimal Discriminator

Derivation Steps

1

Fix generator G, optimize D

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



2

Calculus of variations

$$\partial V / \partial D(x) = 0$$



3

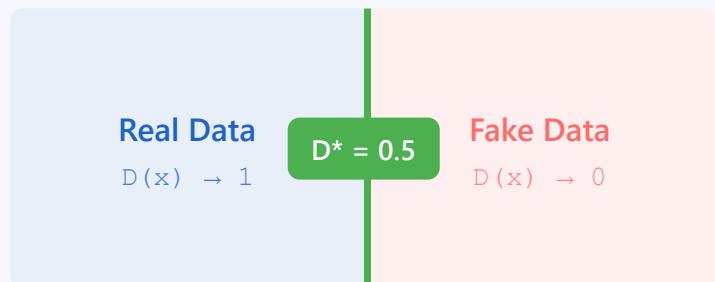
Solve for optimal D

$$p_{\text{data}}(x) / D(x) = p_g(x) / (1 - D(x))$$

Key Insights

- Sigmoid activation natural choice
- Binary cross-entropy loss
- Depends on density ratio
- Neural network approximation

Decision Boundary



At equilibrium: $p_g = p_{\text{data}}$

✓ Optimal Discriminator

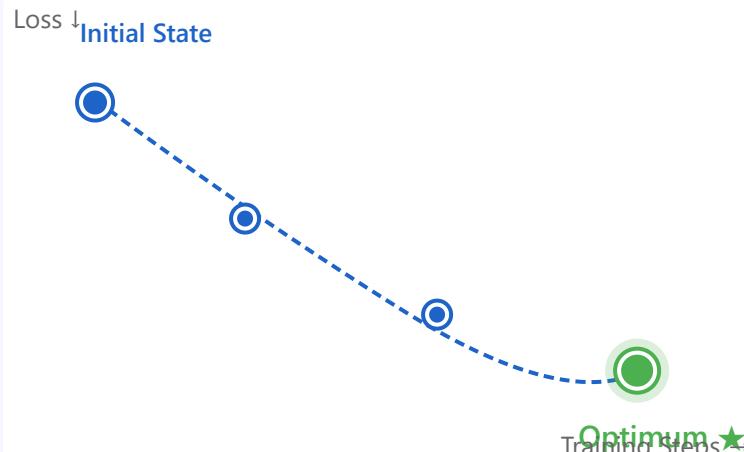
$$D^*(x) = p_{\text{data}}(x) / (p_{\text{data}}(x) + p_g(x))$$

Global Optimum



Global minimum achieved if and only if $p_g = p_{\text{data}}$

Convergence Trajectory



At Global Optimum

Discriminator $D^*(x) = 1/2$

Value Function $v = -\log(4)$

Distributions $p_g = p_{\text{data}}$

JS Divergence Connection

Minimizing GAN objective equals minimizing Jensen-Shannon divergence

$$\text{JS}(p_{\text{data}} \parallel p_g) = 0 \Leftrightarrow p_g = p_{\text{data}}$$

✓ Convergence Properties

- Uniqueness of global optimum
- Proof under ideal conditions
- Theoretical guarantees

⚠ Practical Challenges

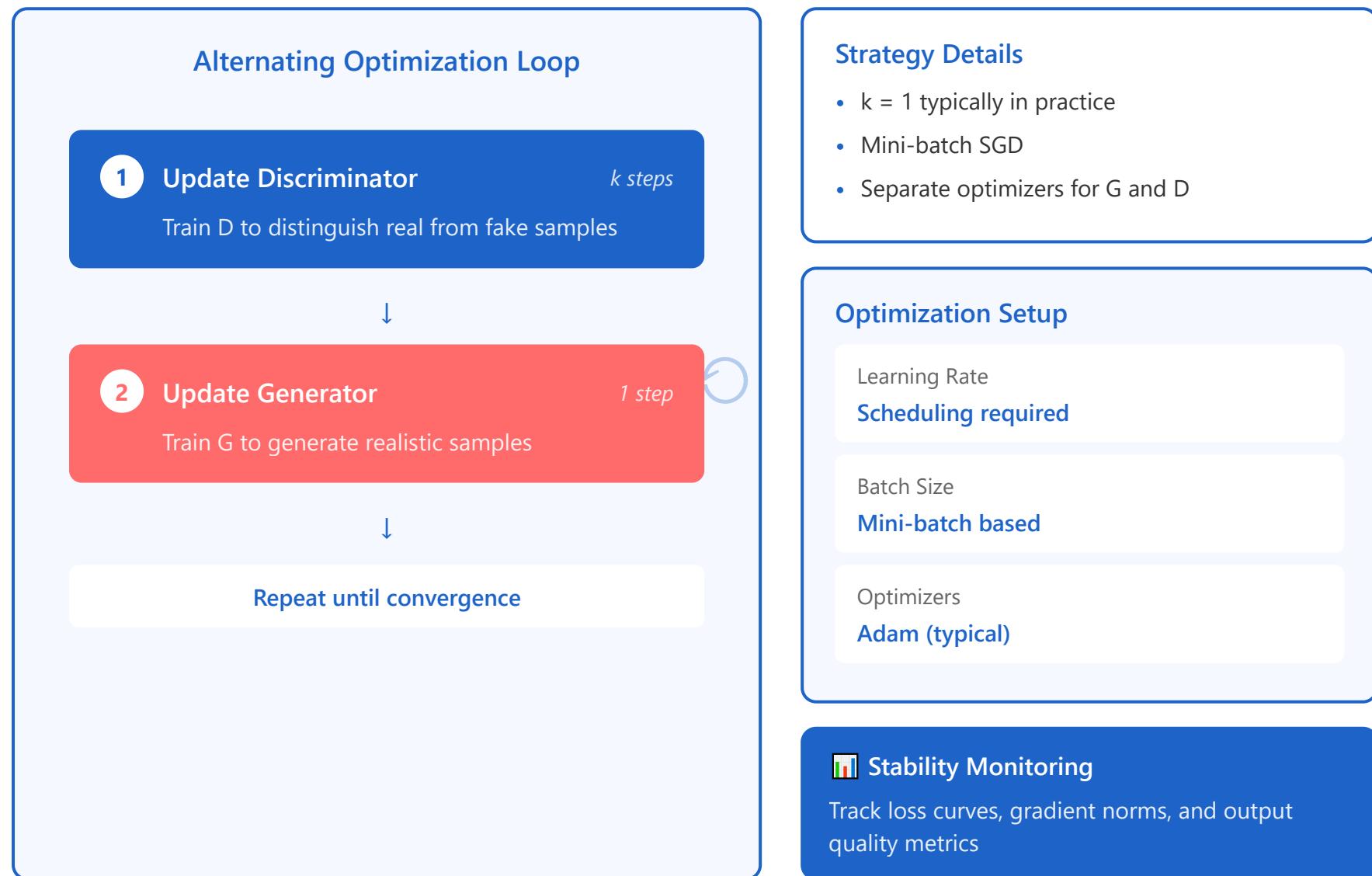
- Mode collapse issues
- Training instability
- Non-convex optimization

Part 3/6:

Training Algorithm

- 9.** Training Process Overview
- 10.** Detailed Algorithm
- 11.** Gradient Flow
- 12.** Non-Saturating Loss
- 13.** Practical Tips

Training Process Overview



Part 3/6: Training Algorithm

Detailed Algorithm

Training Procedure

1 Sample minibatch of m noise samples $\{z^{(1)} \dots z^{(m)}\}$

2 Sample minibatch of m real samples $\{x^{(1)} \dots x^{(m)}\}$

3 Update Discriminator by ascending gradient:

$$\nabla_{\theta_d} [1/m \sum (\log D(x^{(i)}) + \log(1-D(G(z^{(i)}))))]$$

Switch to Generator

4 Sample new minibatch of noise samples

5 Update Generator by descending gradient:

$$\nabla_{\theta_g} [1/m \sum \log(1-D(G(z^{(i)})))]$$

Update Strategy

D Ascending gradient (maximize)

G Descending gradient (minimize)

Notation

m Minibatch size

z Noise vector

x Real sample

θ_d D parameters

θ_g G parameters

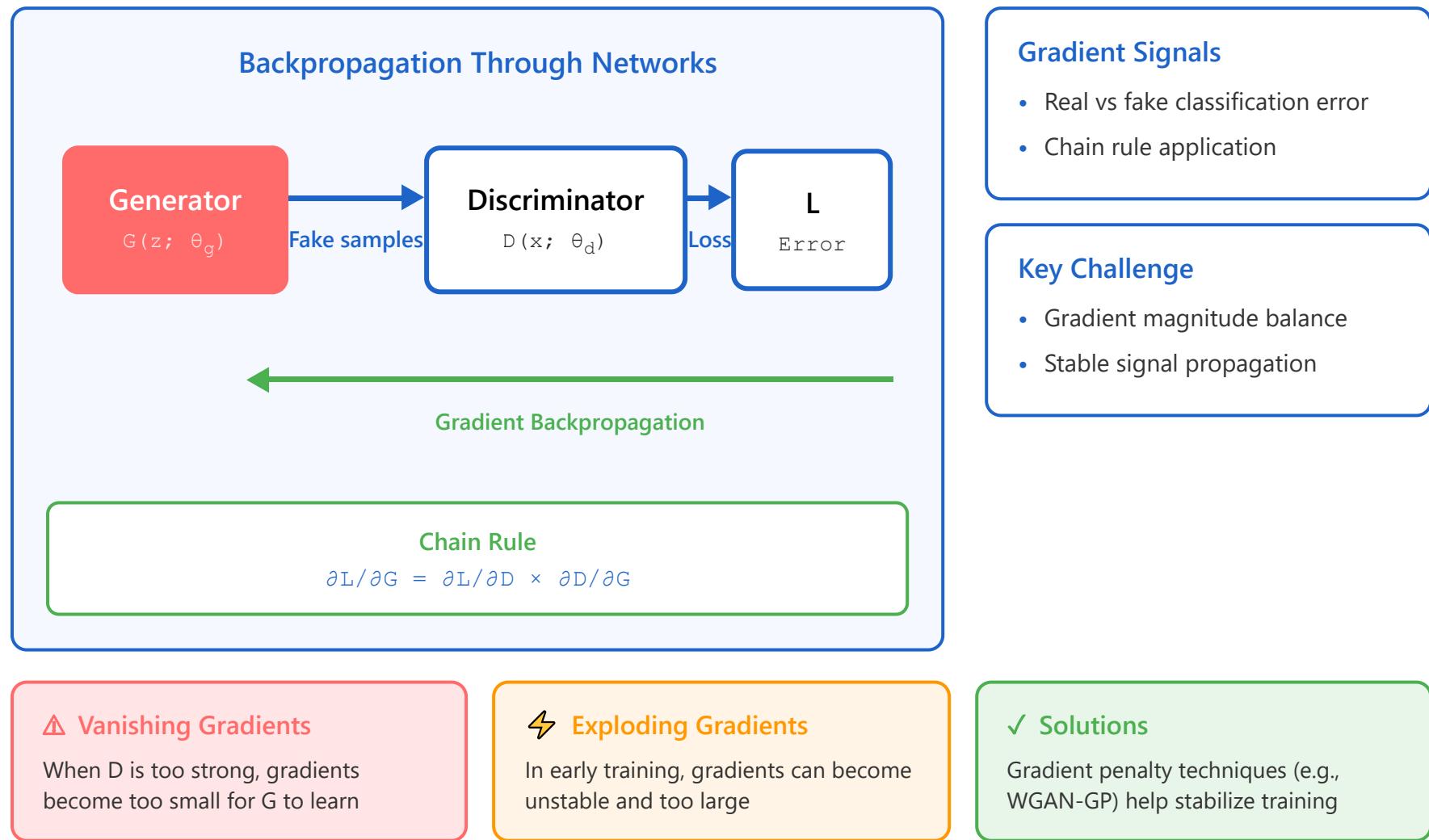
6

Repeat until convergence

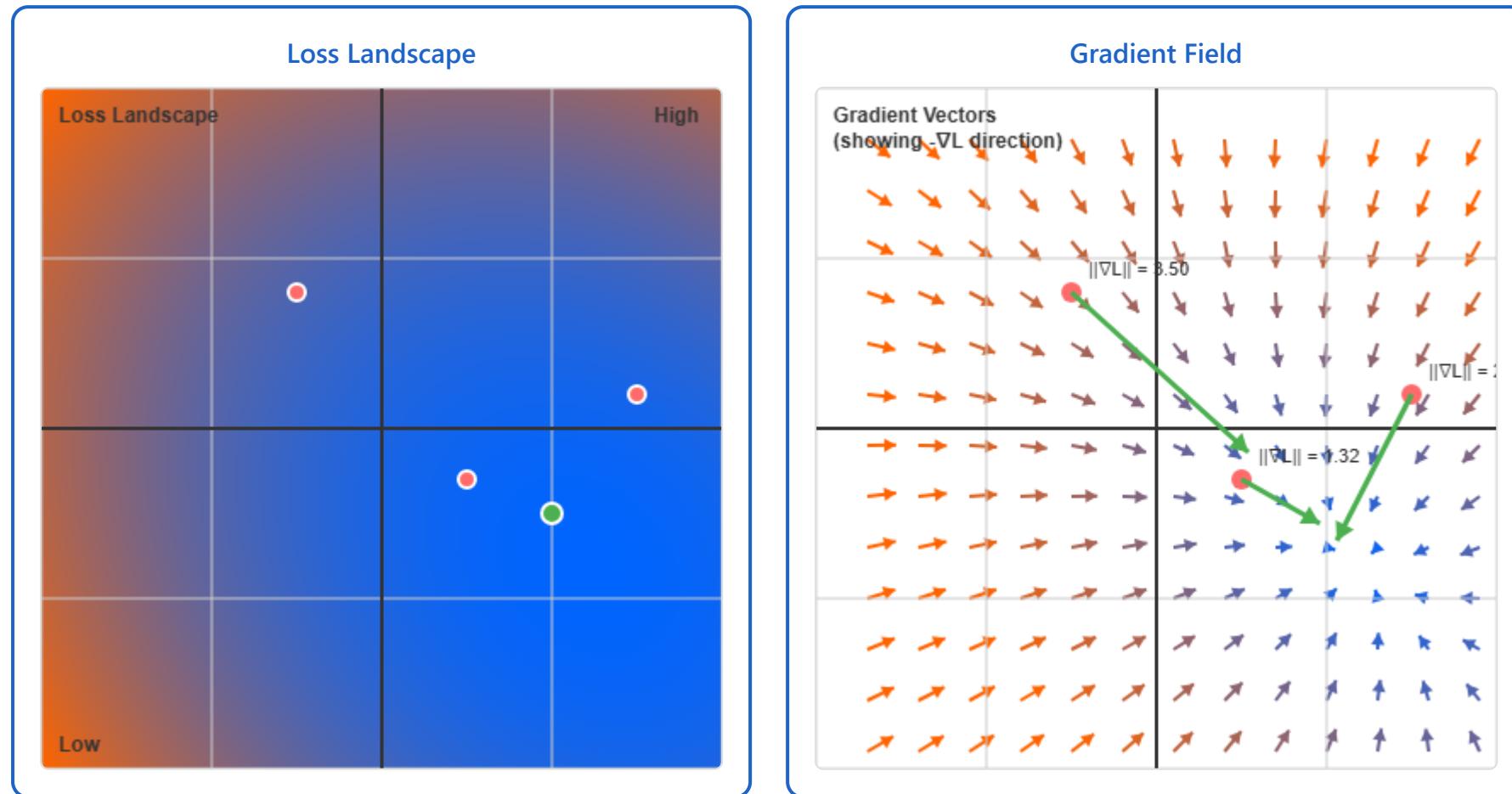


Iterate Until Equilibrium

Gradient Flow



Interactive Gradient Flow Visualization

ResetAdd Sample PointClear Points

Gradient Scale

Normal ▾

Numerical Example: 2D Gradient Computation

Step 1: Define Loss Function

$$L(x, y) = (x - 1)^2 + (y + 0.5)^2 + 0.5 \cdot x \cdot y$$

This represents a simplified discriminator loss in 2D parameter space

Step 2: Compute Partial Derivatives

$$\begin{aligned}\partial L / \partial x &= 2(x - 1) + 0.5 \cdot y \\ \partial L / \partial y &= 2(y + 0.5) + 0.5 \cdot x\end{aligned}$$

Step 3: Example at Point (0.5, -0.3)

$$x = 0.5, y = -0.3$$

$$\begin{aligned}\partial L / \partial x &= 2(0.5 - 1) + 0.5 \cdot (-0.3) = -1.0 - 0.15 = -1.15 \\ \partial L / \partial y &= 2(-0.3 + 0.5) + 0.5 \cdot (0.5) = 0.4 + 0.25 = 0.65\end{aligned}$$

Gradient Vector: $\nabla L = (-1.15, 0.65)$

Magnitude: $\|\nabla L\| = \sqrt{(-1.15)^2 + 0.65^2} \approx 1.32$

Step 4: Parameter Update (Gradient Descent)

Learning rate: $\alpha = 0.1$

$$\begin{aligned}x_{\text{new}} &= x - \alpha \cdot (\partial L / \partial x) = 0.5 - 0.1 \cdot (-1.15) = 0.615 \\ y_{\text{new}} &= y - \alpha \cdot (\partial L / \partial y) = -0.3 - 0.1 \cdot (0.65) = -0.365\end{aligned}$$

Updated Position: (0.615, -0.365)

Movement towards minimum: Point moves closer to (1, -0.5)

Gradient Magnitude Impact

- $\|\nabla L\| < 0.01$: Vanishing
- $0.01 \leq \|\nabla L\| \leq 10$: Normal

Backprop Through Generator

$\partial L / \partial \theta_g = \partial L / \partial x \cdot \partial x / \partial \theta_g$
Chain rule multiplies gradients

- $\|\nabla L\| > 10$: Exploding
- Affects learning speed

Deep networks: many multiplications
Can cause vanishing/exploding

Critical Insights

- Gradient direction: steepest ascent
- Negative gradient: descent
- Zero gradient: local min/max
- Balance is crucial for GANs

Real GAN Scenario

If D too good: $\partial L / \partial D \rightarrow 0$
Then: $\partial L / \partial G \rightarrow 0$ (vanishing)
G cannot learn effectively
Needs balanced training

Part 3/6: Training Algorithm

Practical Tips

1 Data & Activation

- ✓ Normalize inputs to `[-1, 1]`
- ✓ Use `tanh` activation for generator output
- ✓ `LeakyReLU` in D, `ReLU` in G

2 Normalization & Regularization

- ✓ Batch normalization (except last G, first D layer)
- ✓ Label smoothing: `real=0.9, fake=0.1`
- ✓ Add noise to discriminator inputs

3 Optimization

- ✓ Use Adam optimizer with $\beta_1 = 0.5$

Architecture Guide

Generator

- Dense/Conv Layers
- ReLU
- Batch Norm
- ...
- No BN (Last Layer)
- tanh Output

Discriminator

- No BN (First Layer)
- LeakyReLU
- Batch Norm
- ...
- LeakyReLU
- Sigmoid Output

⚠ Monitoring

Watch D loss carefully - if it drops to 0, G won't learn properly

✓ Monitor D loss - shouldn't go to 0

✓ Recommended Optimizer

Adam with $\beta_1 = 0.5$, $\beta_2 = 0.999$

Part 3/6: Training Algorithm

Practical Tips

1 Data & Activation

- ✓ Normalize inputs to `[-1, 1]`
- ✓ Use `tanh` activation for generator output
- ✓ `LeakyReLU` in D, `ReLU` in G

2 Normalization & Regularization

- ✓ Batch normalization (except last G, first D layer)
- ✓ Label smoothing: `real=0.9, fake=0.1`
- ✓ Add noise to discriminator inputs

3 Optimization

- ✓ Use Adam optimizer with $\beta_1 = 0.5$

Architecture Guide

Generator

- Dense/Conv Layers
- ReLU
- Batch Norm
- ...
- No BN (Last Layer)
- tanh Output

Discriminator

- No BN (First Layer)
- LeakyReLU
- Batch Norm
- ...
- LeakyReLU
- Sigmoid Output

⚠ Monitoring

Watch D loss carefully - if it drops to 0, G won't learn properly

✓ Recommended Optimizer

✓ Monitor D loss - shouldn't go to 0

Adam with $\beta_1 = 0.5$, $\beta_2 = 0.999$

Interactive Practice

Try GAN Lab →

Part 4/6:

Key Challenges

- 14.** Mode Collapse
- 15.** Training Instability
- 16.** Evaluation Difficulties
- 17.** Vanishing Gradient Details
- 18.** Common Failure Patterns

Part 4/6: Key Challenges

Mode Collapse

Output Diversity Comparison

✓ Healthy: Diverse Outputs



△ Partial Collapse



X Complete Collapse



Problem Description

- Limited variety of samples
- Entire noise → few modes
- Poor gradient signals

Types of Collapse

Partial Collapse
Missing some data modes

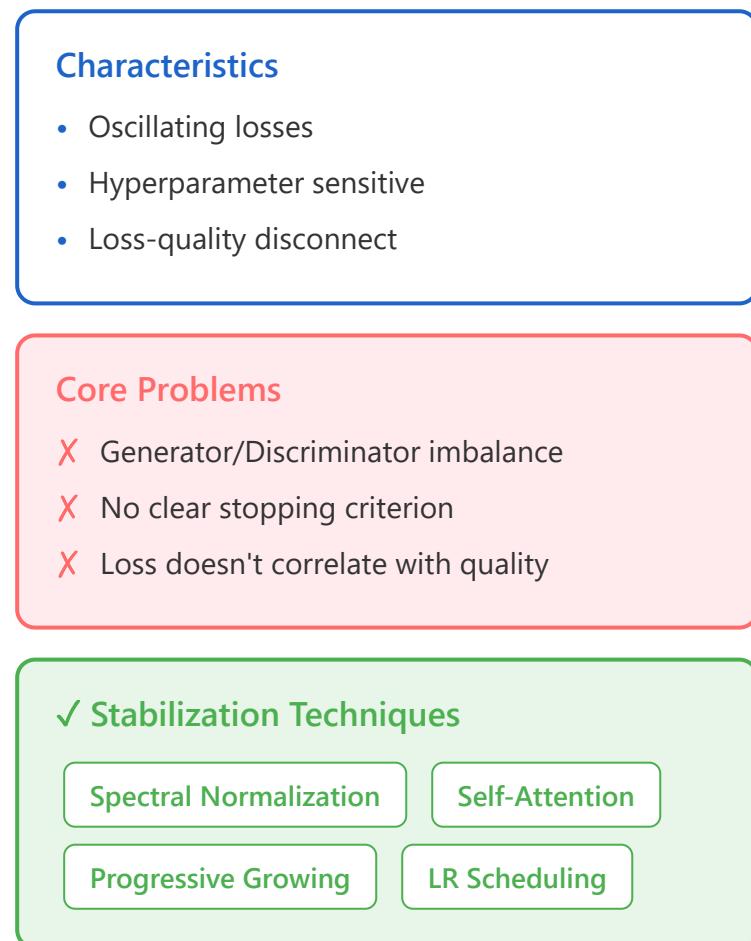
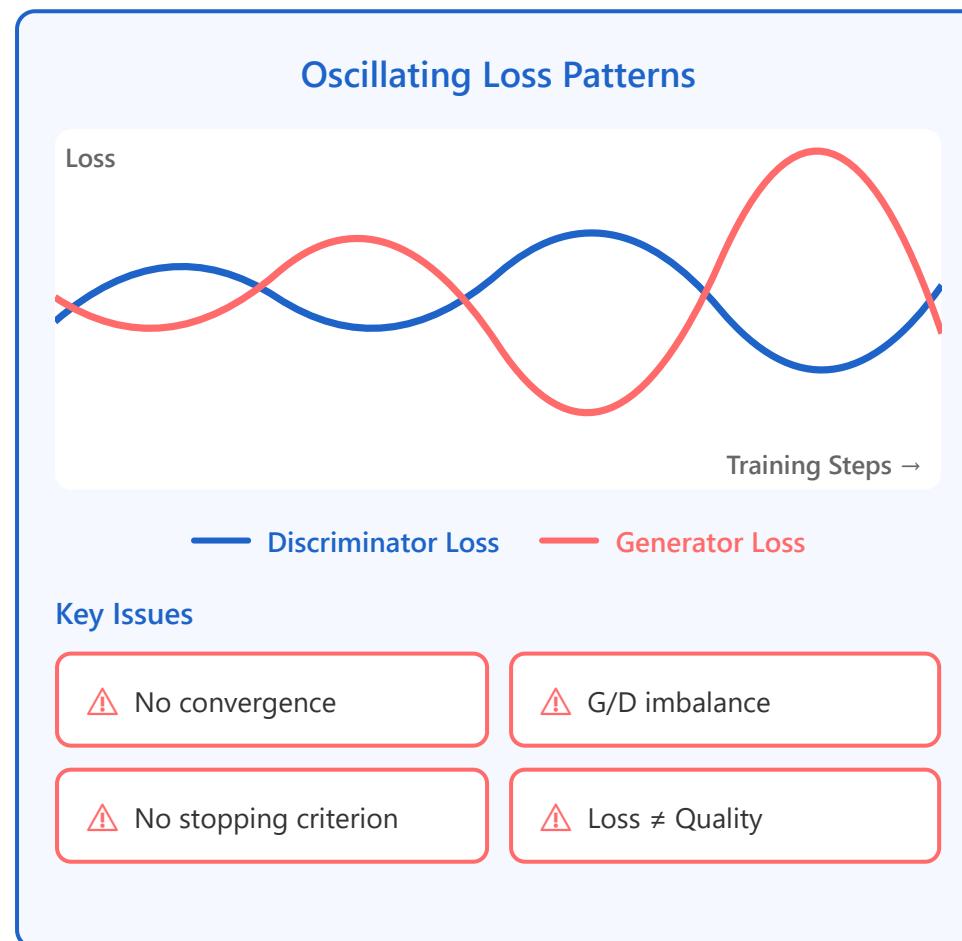
Complete Collapse
Single output regardless of input

Solutions

Minibatch Discrimination Unrolled GANs

Feature Matching Diversity Penalties

Training Instability



Part 4/6: Key Challenges

Evaluation Difficulties



No single metric captures generation quality completely

Metric	What It Measures	Quality	Diversity
Inception Score (IS)	Quality and diversity via classifier confidence	High	High
Fréchet Inception Distance (FID)	Distribution similarity between real and generated	High	High
Precision	Quality of generated samples (realism)	High	Low
Recall	Coverage of real data modes	Low	High
Human Evaluation	Subjective assessment by human raters	High	High



Human evaluation remains the most reliable method despite being time-



Mode coverage assessment is difficult - metrics may miss partial collapse



Computational Cost

Most metrics require running large pre-trained models, making evaluation

consuming and subjective

expensive

Vanishing Gradient Details

Problem Cascade

1

Discriminator becomes too successful



2

D correctly classifies fake samples with high confidence

$$D(G(z)) \rightarrow 0$$



3

Original loss function saturates

$$\log(1 - D(G(z))) \rightarrow \log(1) = 0$$



4

Gradients vanish - Generator receives no learning signal

$$\nabla_G \rightarrow 0$$

When It Occurs

- ⚠ D is too successful
- ⚠ $D(G(z))$ approaches 0
- ⚠ No learning signal for G

Key Insight

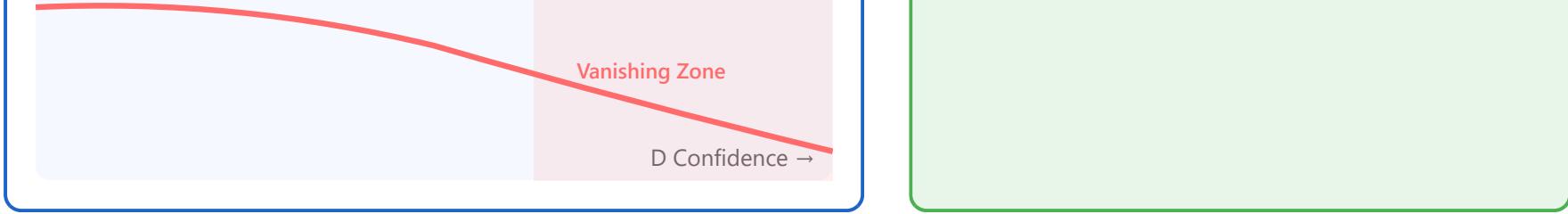
Careful capacity balance between G and D is crucial to prevent one from dominating

✓ Solutions

- Non-saturating loss
- WGAN (Wasserstein GAN)
- Gradient penalty methods
- Balance G and D capacity

Gradient Magnitude Over Training

Gradient Magnitude



Part 4/6: Key Challenges

Common Failure Patterns

VISUAL

Checkerboard Artifacts



Grid-like patterns appear in generated images due to deconvolution operations

Image Quality

TEXTURE

Inconsistent Transitions



Mix of blurry and sharp regions without logical consistency

Coherence

COLOR

Unrealistic Colors



Color distributions don't match real data, often oversaturated or unnatural

Distribution

ANATOMY

Anatomical Errors



Incorrect features in faces, hands, or body proportions

Realism

CRITICAL

Training Collapse After Initial Success



Model performance degrades suddenly after showing good results, often unrecoverable without restart

Training Dynamics



Visual Artifacts

- Checkerboard patterns
- Color inconsistencies



Video Issues

Temporal inconsistency between frames in video generation tasks



Resolution Limits

Quality degrades without progressive growing techniques

- Blurry/sharp mixing

Part 5/6:

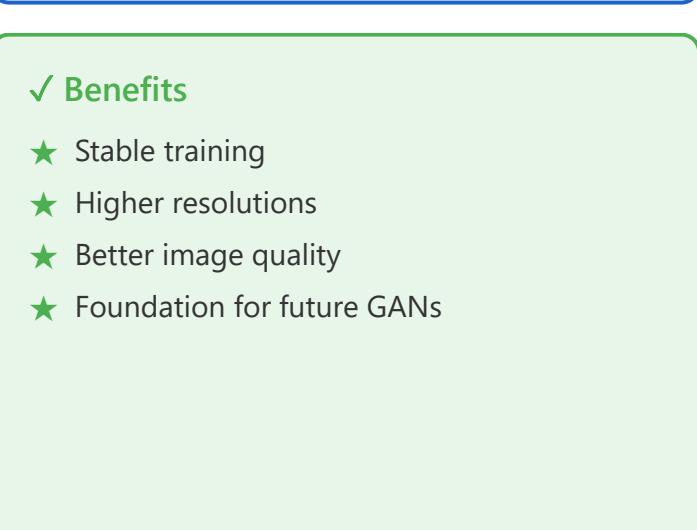
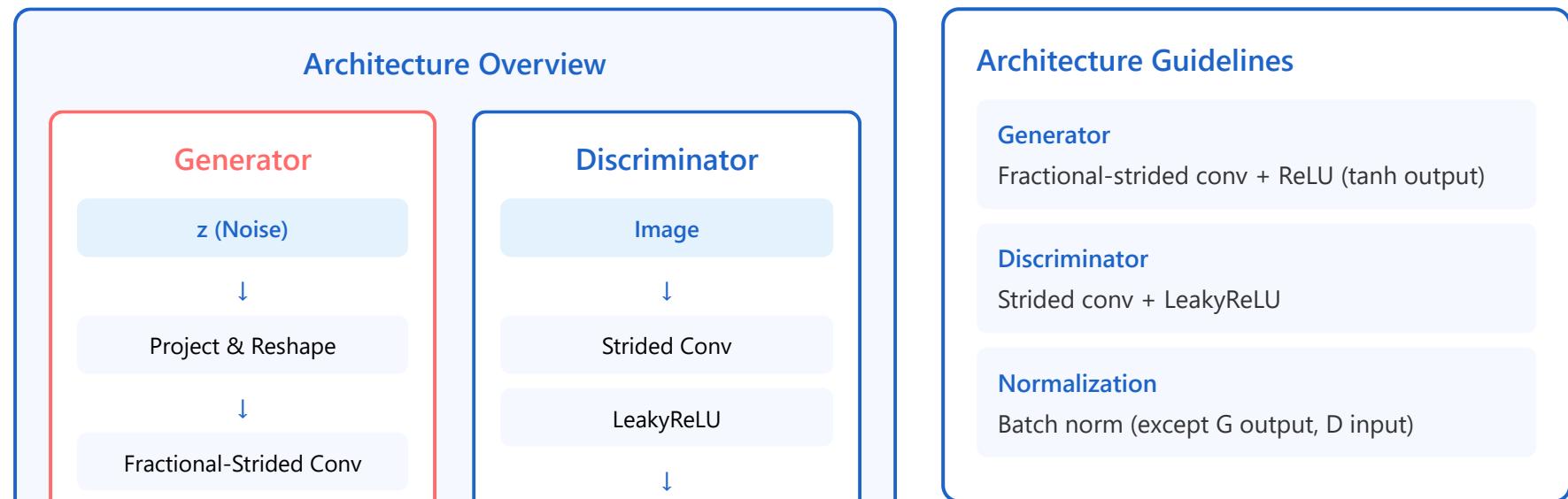
Improvement Techniques

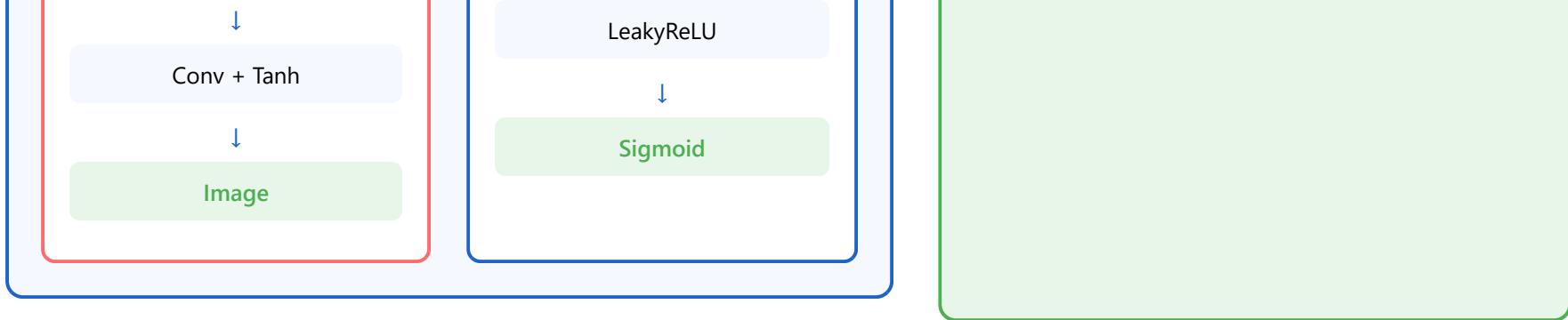
- 19.** DCGAN Architecture
- 20.** Wasserstein GAN
- 21.** Conditional GAN
- 22.** Other Variants

Part 5/6: Improvement Techniques

DCGAN (2015)

Deep Convolutional GAN





Key Features

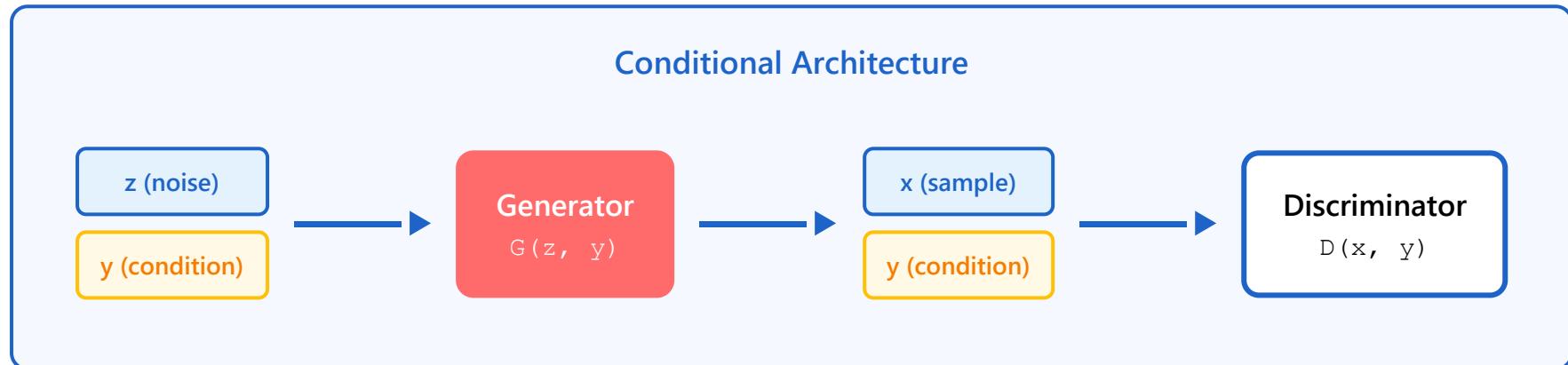
- ✓ Replace fully connected layers with convolutions
- ✓ No pooling layers - use strided convolutions
- ✓ Batch normalization in both G and D

Original Paper

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

Radford et al., 2015

Conditional GAN (cGAN)



Key Idea: Both G and D are conditioned on additional information y

Conditioning Types

y can be: class labels, text descriptions, images, or any structured information

Core Concept

Condition generation on additional information to enable controllable and targeted output

Applications

- Class-conditional generation
- Image-to-image (Pix2Pix)
- Text-to-image synthesis
- Super-resolution

✓ Key Benefit

Controllable generation capabilities
- direct control over output characteristics

 [Original Paper](#)

Conditional Generative Adversarial Nets

Mehdi Mirza, Simon Osindero (2014)

 [View on arXiv](#)

Part 5/6: Improvement Techniques

Wasserstein GAN (WGAN)

ORIGINAL Standard GAN

Distance Metric

Jensen-Shannon divergence

$$\text{JS}(\mathbf{p}_{\text{data}} \parallel \mathbf{p}_g)$$

Output Layer

Sigmoid (Discriminator)

Constraint

No explicit constraint

Training

Unstable, oscillating losses

WGAN Wasserstein GAN

Distance Metric

Wasserstein distance (Earth Mover)

$$W(\mathbf{p}_{\text{data}}, \mathbf{p}_g)$$

Output Layer

Linear (Critic, no sigmoid)

Constraint

Lipschitz constraint (weight clipping)

Training

Stable, meaningful loss

VS



Stable Training

More consistent convergence dynamics



Meaningful Loss

Loss correlates with quality



Mode Coverage

Better distribution coverage



WGAN-GP Enhancement

Gradient penalty replaces weight clipping for better Lipschitz constraint enforcement

Original Papers

WGAN

Wasserstein GAN

Arjovsky et al., 2017

WGAN-GP

Improved Training of Wasserstein GANs

Gulrajani et al., 2017

Part 5/6: Improvement Techniques

Other Improvements



ProGAN

2017

Progressive growing of resolution for high-quality image generation

Resolution Scaling



SAGAN

2018

Self-attention mechanisms for long-range dependencies

Attention



CycleGAN

2017

Unpaired image-to-image translation with cycle consistency

Unpaired Translation



StyleGAN

2018

Style-based generator architecture with unprecedented control

Style Control



BigGAN

2018

Large-scale training with architectural tricks and optimization

Scale



GauGAN

2019

NVIDIA's semantic image synthesis from layout sketches

Semantic Control



Continuous Evolution



Alternative Approach

Each variant addresses specific limitations and pushes boundaries in different directions: resolution, quality, control, and applicability

Diffusion models emerged as a powerful alternative, offering stable training and high-quality generation

Part 6/6:

Hands-on & Applications

- 23. Hands-on Guide**
- 24. Applications and Wrap-up**

Lecture15_31 Placeholder

콘텐츠 준비 중입니다.

Lecture15_32 Placeholder

콘텐츠 준비 중입니다.

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com