Lecture 12.4: GenAI: Generative Adversarial Networks

The Art of Adversarial Training

Heman Shakeri

Generative Adversarial Net

The Paper That Started It All

"Generative Adversarial Nets" (2014) Ian Goodfellow et al.
`https://arxiv.org/abs/1406.2661`
Accepted at NIPS 2014

- Train two networks that compete against each other!
- Now an old idea!! But back the Yann LeCun called GANs "the most interesting idea in the last 10 years in ML".
- First to generate high-resolution images

The Adversarial Game

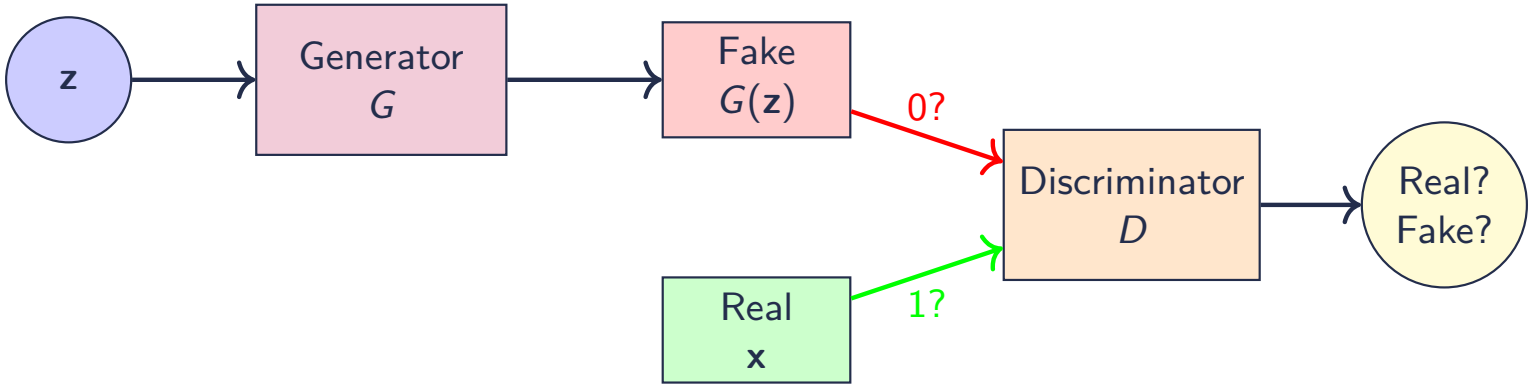
Two players in a zero-sum game:

Generator G

- Creates fake data
- Tries to fool discriminator
- Learns data distribution

Discriminator D

- Judges real vs fake
- Tries to catch generator
- Learns decision boundary



The Mathematical Objective

Minimax game:

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

Breaking it down:

- $D(\mathbf{x})$: Discriminator's output for real data (want $\to 1$)
- $D(G(\mathbf{z}))$: Discriminator's output for fake data (want $\to 0$)

Discriminator maximizes:

- $\log D(\mathbf{x})$ high when real data detected correctly
- $\log(1 - D(G(\mathbf{z})))$ high when fake data detected correctly

Generator minimizes:

- $\log(1 - D(G(\mathbf{z})))$ low when fooling discriminator
- Equivalently: maximize $\log D(G(\mathbf{z}))$ (non-saturating version)

Training Algorithm: The Dance

Alternating optimization:

1. **Train Discriminator** (fix Generator)
   - Sample real data $\mathbf{x} \sim p_{data}$
   - Sample noise $\mathbf{z} \sim p_z$, generate fake $G(\mathbf{z})$
   - Update D to maximize $\log D(\mathbf{x}) + \log(1 - D(G(\mathbf{z})))$
2. **Train Generator** (fix Discriminator)
   - Sample noise $\mathbf{z} \sim p_z$
   - Update G to maximize $\log D(G(\mathbf{z}))$

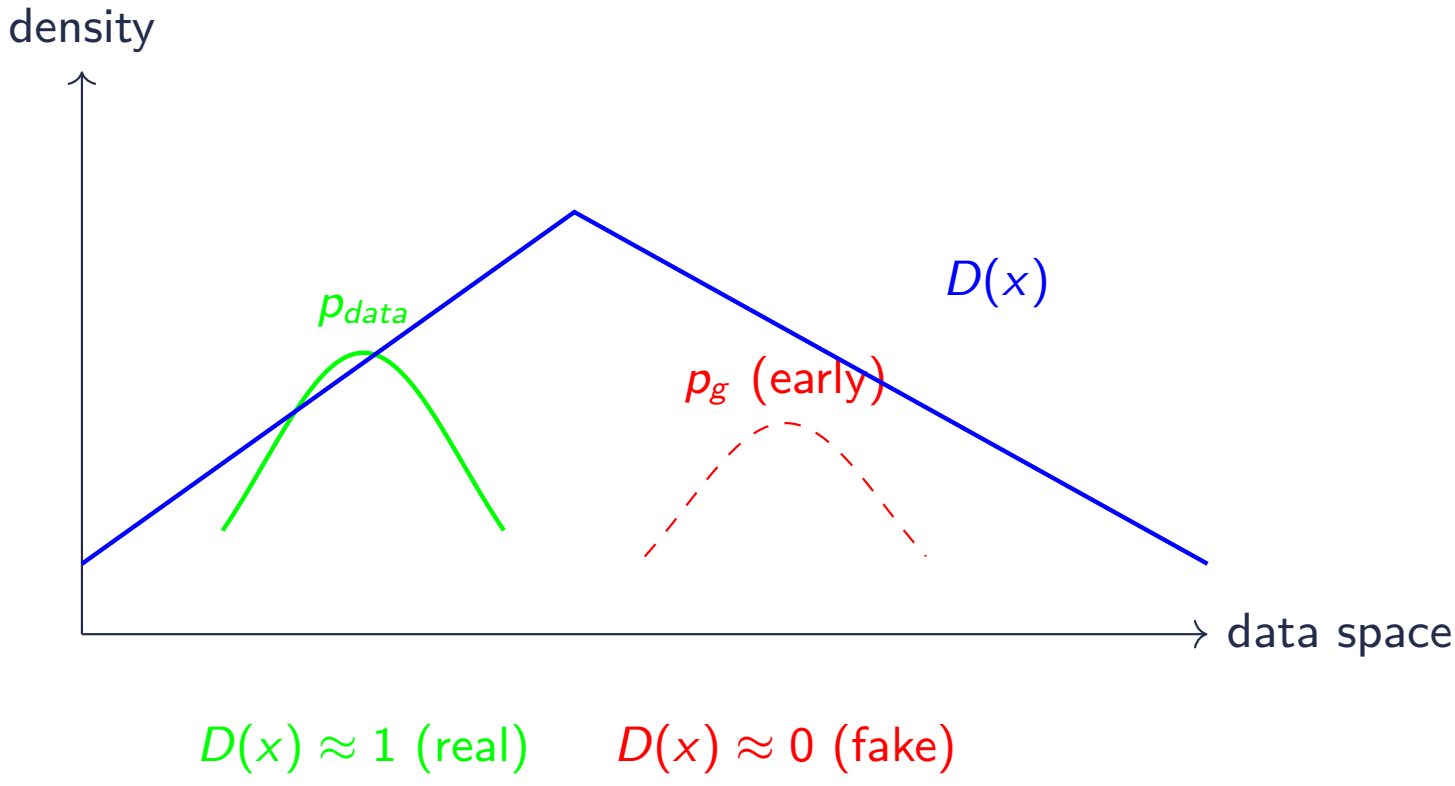
Key insight: Don't train to equilibrium! Alternate frequently

Typically: Train D for k steps, then train G for 1 step

$k = 1$ often works, but $k = 5$ common for stability

Visualizing the Training Process

Early training: Discriminator easily distinguishes real from fake

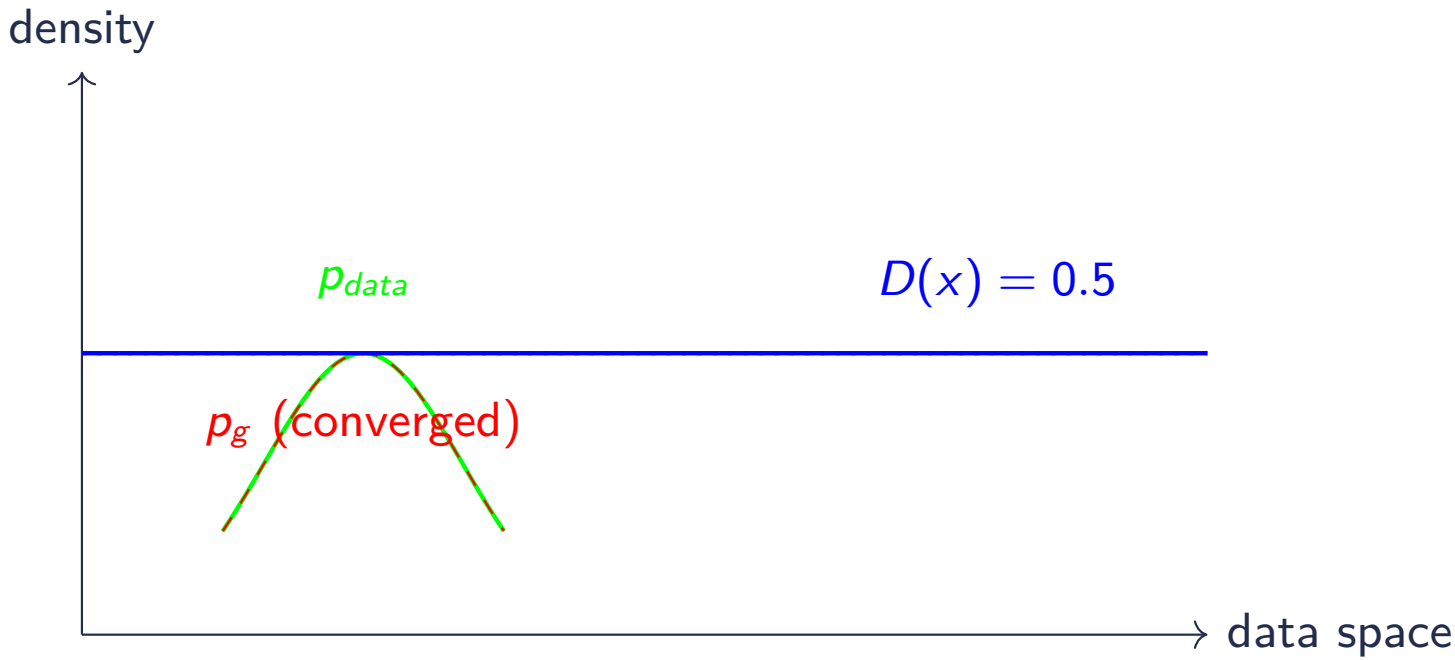


Discriminator's job: Output 1 for real, 0 for fake

When D is confident, G gets strong gradient signal to improve

Nash Equilibrium: When GANs Converge

Convergence: Generator matches data distribution perfectly



D cannot tell real from fake!

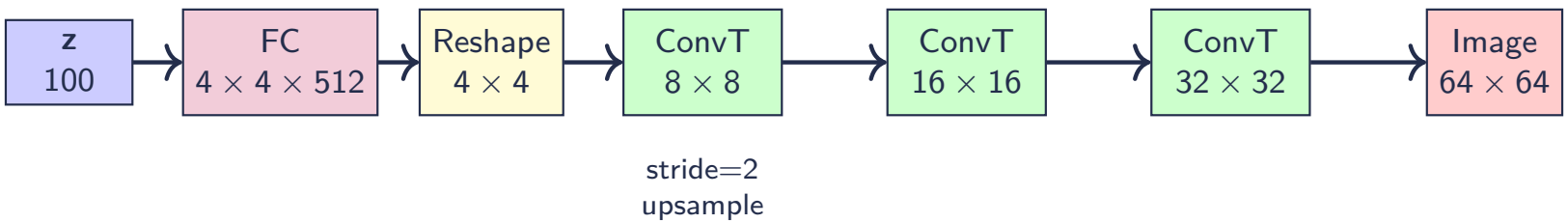
Why $D(x) = 0.5$ is optimal:

- When $p_g = p_{data}$, every sample is equally likely to be real or fake
- Discriminator has no information to distinguish them
- $D(x) = 0.5$ everywhere means "I'm guessing randomly"
- This is the **Nash equilibrium** — neither player can improve!

Generator Architecture

Task: Transform noise $\mathbf{z}$ into realistic data $G(\mathbf{z})$

For images: Use transposed convolutions (Lecture 6!)



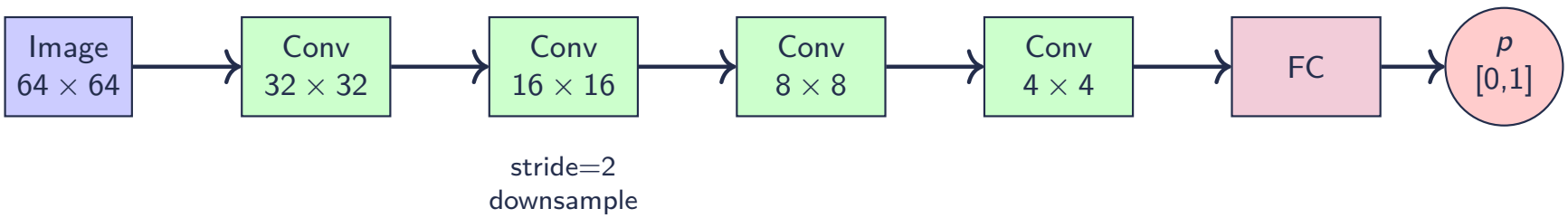
Key techniques:

- Batch normalization for stability
- ReLU activations (hidden), Tanh (output)
- No pooling, use strided convolutions

Discriminator Architecture

Task: Classify real vs. fake $D(\mathbf{x}) \in [0, 1]$

For images: Standard CNN classifier (Lecture 4-5!)



Key techniques:

- Leaky ReLU activations
- Strided convolutions instead of pooling
- Sigmoid output for binary classification

The Training Instability Problem

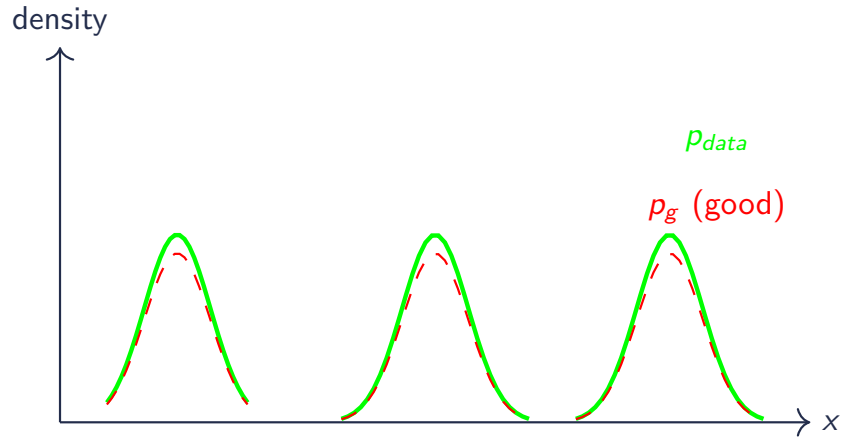
GANs are notoriously hard to train!

Common failure modes:

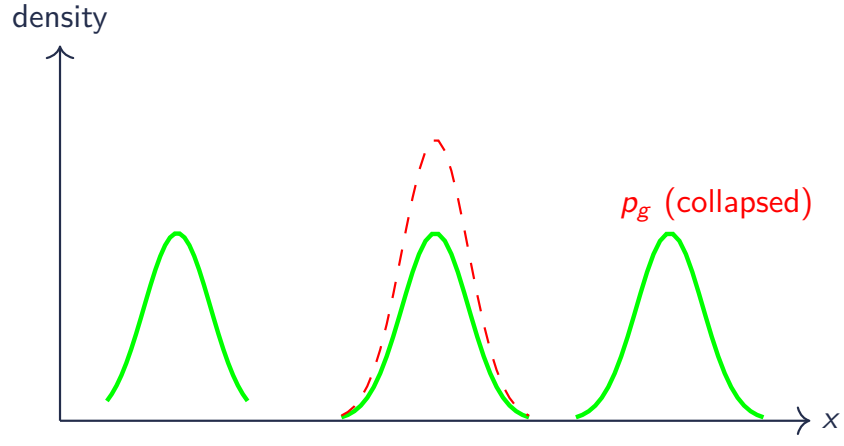
1. **Mode collapse:** G generates only a few types of samples
   - Finds a few images that fool D, ignores diversity
   - Missing entire modes of data distribution
2. **Vanishing gradients:** D too good too fast
   - When D is perfect, G gets no gradient signal
   - $\log(1 - D(G(\mathbf{z}))) \approx 0$ everywhere
3. **Oscillation:** No convergence, just cycling
   - G and D chase each other endlessly
   - Never reach Nash equilibrium

Mode Collapse Illustration

Ideal: Generator captures all modes of data distribution



Mode collapse: Generator only captures subset



Why Training is Hard: The Gradient Problem

Generator's gradient:

$$\nabla_{\theta_g} \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

Problem: When D is confident (early training), gradient vanishes!

- If $D(G(\mathbf{z})) \approx 0$, then $\log(1 - D(G(\mathbf{z}))) \approx 0$
- Flat region $\rightarrow$ tiny gradients $\rightarrow$ slow learning

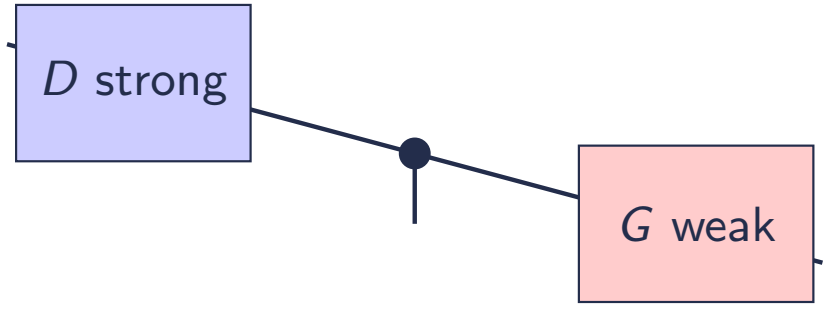
Solution: Use non-saturating loss

$$\max_G \mathbb{E}_{\mathbf{z}}[\log D(G(\mathbf{z}))]$$

Instead of minimizing probability of being fake, maximize probability of being real!

This provides better gradients early in training when D is strong

The Balancing Act



Bad: No gradient for G

Good: Balanced training

The challenge: Keep them balanced throughout training!

Improvements: DCGAN (2015)

"Deep Convolutional GAN" — architectural guidelines

Key contributions:

1. Replace pooling with strided convolutions
2. Use batch normalization (except output of G and input of D)
3. Remove fully connected hidden layers
4. Use ReLU in G (except output: Tanh)
5. Use LeakyReLU in D

Result: Much more stable training!

These became the standard architecture guidelines

DCGAN showed GANs could learn hierarchical representations

Deep networks with proper architecture work much better than shallow ones

Other Important Improvements

Spectral Normalization (2018):

- Normalize weights by their spectral norm
- Enforces Lipschitz constraint more elegantly
- Very stable, widely adopted

Progressive Growing (2017):

- Start with low resolution, gradually increase
- Train 4×4, then 8×8, then 16×16, etc.
- Enabled high-resolution generation (1024×1024)

StyleGAN (2018-2019):

- Control generation at different scales
- Learns disentangled representations
- State-of-the-art quality for faces

CycleGAN: Unpaired Translation

Problem: What if we don't have paired training data?

Can we learn horse $\leftrightarrow$ zebra without matched pairs?

Solution: Cycle consistency!

- Learn two mappings: $G : X \to Y$ and $F : Y \to X$
- Enforce $F(G(\mathbf{x})) \approx \mathbf{x}$ and $G(F(\mathbf{y})) \approx \mathbf{y}$

Loss:

$$\mathcal{L} = \mathcal{L}_{GAN}(G, D_Y) + \mathcal{L}_{GAN}(F, D_X) + \lambda \mathcal{L}_{cyc}(G, F)$$

Cycle consistency loss:

$$\mathcal{L}_{cyc} = \mathbb{E}_{\mathbf{x}}[\|F(G(\mathbf{x})) - \mathbf{x}\|_1] + \mathbb{E}_{\mathbf{y}}[\|G(F(\mathbf{y})) - \mathbf{y}\|_1]$$

Enables amazing applications: photo $\leftrightarrow$ painting, summer $\leftrightarrow$ winter, etc.

GANs vs VAEs vs Diffusion

	GANs	VAEs	Diffusion
Quality	Sharp, realistic	Blurry	Sharp, realistic
Diversity	Mode collapse risk	Good coverage	Excellent coverage
Training	Unstable, tricky	Stable, easy	Stable, slower
Likelihood	No explicit $p(x)$	Bounded $\log p(x)$	Tractable $p(x)$
Speed	Fast sampling	Fast sampling	Slow sampling
Control	Good (cGAN)	Good (cVAE)	Excellent (guidance)

The current state (2025):

- Diffusion models dominate high-quality image generation
- VAEs used for compression (Stable Diffusion)
- GANs still used for fast inference, specific domains

Why Diffusion Won

GANs' Achilles Heels:

1. Training instability makes scaling difficult
2. Mode collapse hurts diversity
3. No explicit likelihood, hard to evaluate
4. Difficult to condition precisely

Diffusion's Advantages:

1. Stable training, scales reliably
2. Excellent coverage of data distribution
3. Principled probabilistic framework
4. Natural conditioning through cross-attention
5. Superior controllability (guidance, inpainting)

The verdict: By 2022-2023, diffusion models overtook GANs

DALL-E 2, Imagen, Stable Diffusion all use diffusion, not GANs

GANs' Lasting Impact

Despite being "dethroned," GANs revolutionized the field:

Conceptual contributions:

- Adversarial training as a principle
- Learning from implicit feedback
- Game-theoretic view of learning

Technical contributions:

- Architectural innovations (DCGAN, StyleGAN)
- Perceptual losses and adversarial losses
- Image-to-image translation frameworks

Still used today:

- Super-resolution (ESRGAN)
- Fast inference applications
- Video generation (some models)
- Discriminator as loss function for other models

Practical Tips for Training GANs

If you must train a GAN:

1. Start with proven architecture (DCGAN guidelines)
2. Use appropriate learning rates (~ 0.0002 with Adam)
3. Train discriminator more frequently ($k = 5$ steps)
4. Monitor both losses, look for balance
5. Use techniques from WGAN-GP or Spectral Norm
6. Generate samples frequently to check for mode collapse
7. Consider using pre-trained discriminators
8. Be patient, expect some instability

Or just use diffusion models!

Much more stable and better results in 2025