Generative Adversarial Networks

Handout Notes - Generative and Deep Learning (GDL)

Davide Bacciu - University of Pisa

Notation. Observed data are denoted by $\mathbf{X}$ and samples by $\mathbf{x} \in \mathbb{R}^D$. Latent random variables are denoted by $\mathbf{Z}$ and latent samples by $\mathbf{z} \in \mathbb{R}^K$. A dataset is $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ with $N = |\mathcal{D}|$. The generator is denoted by G_{θ_G} and the discriminator (or critic) by D_{θ_D}. The latent prior is $p(\mathbf{z})$, usually simple, such as $\mathcal{N}(\mathbf{0}, I)$. The generator induces a model distribution $p_{\theta_G}(\mathbf{x})$ implicitly through the sampling process

$$\mathbf{z} \sim p(\mathbf{z}), \qquad \tilde{\mathbf{x}} = G_{\theta_G}(\mathbf{z}).$$

We use $p_{\text{data}}(\mathbf{x})$ for the true data distribution. Expectations are written $\mathbb{E}_{\mathbf{x}\sim p}[\cdot]$.

1 From explicit density learning to implicit generation

Generative modeling can be approached in two broad ways. In an *explicit* approach, one tries to learn a density model

$$p_\theta(\mathbf{x})$$

or a tractable approximation to it. Variational autoencoders are a canonical example: they introduce latent variables, define a probabilistic decoder, and optimize a lower bound on log-likelihood.

Generative adversarial networks take a different path. Instead of explicitly modeling a density, they learn a *sampling process*. The goal is not primarily to compute $p_\theta(\mathbf{x})$, but to generate samples whose distribution is close to the data distribution. This is why GANs are called *implicit* generative models.

This distinction is conceptually important. A VAE answers the question

> *Can we define and approximately optimize a probabilistic model of the data?*

A GAN instead answers

> *Can we train a neural network to transform simple random noise into realistic data samples?*

The GAN viewpoint is attractive because it avoids explicit likelihood computation in high-dimensional spaces, which is often difficult or restrictive. But it also removes the stabilizing structure of explicit density learning and replaces it with a game between two networks.

2 Learning to sample

The fundamental idea behind GANs is deceptively simple.

Suppose we know how to sample from a very simple latent prior, for example

$$\mathbf{z} \sim p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, I).$$

A generator network transforms this latent noise into a synthetic sample:

$$\tilde{\mathbf{x}} = G_{\theta_G}(\mathbf{z}).$$

This induces a model distribution on the output space, even if we cannot write that distribution in closed form.

Thus the problem becomes:

How do we train G_{θ_G} so that the distribution of generated samples matches p_{data}?

There is no direct supervised target for G. For a real data point $\mathbf{x}$, we do not know which latent vector $\mathbf{z}$ should map to it. GANs solve this by introducing a second network that evaluates the quality of generated samples.

3 Adversarial learning: generator and discriminator

A GAN consists of two models trained simultaneously:

- a **generator** G_{θ_G}, which maps random noise to synthetic samples;
- a **discriminator** D_{θ_D}, which takes an input sample and outputs a score interpreted as the probability that the sample is real.

The discriminator receives either:

- a real sample $\mathbf{x} \sim p_{\text{data}}$,
- or a generated sample $\tilde{\mathbf{x}} = G_{\theta_G}(\mathbf{z})$ with $\mathbf{z} \sim p(\mathbf{z})$.

Its goal is to distinguish real from fake:

$$D_{\theta_D}(\mathbf{x}) \to 1, \qquad D_{\theta_D}(G_{\theta_G}(\mathbf{z})) \to 0.$$

The generator, in contrast, wants the discriminator to classify generated samples as real:

$$D_{\theta_D}(G_{\theta_G}(\mathbf{z})) \to 1.$$

This sets up a two-player minimax game. The generator tries to fool the discriminator, while the discriminator tries not to be fooled.

4 The original GAN objective

The classical GAN objective is

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

with

$$V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))].$$

The discriminator maximizes this objective, because it wants:

- $\log D(\mathbf{x})$ to be large on real data,
- $\log(1 - D(G(\mathbf{z})))$ to be large on fake data.

The generator minimizes the same objective, because it wants generated samples to make the second term small.

Worked example — Interpreting the GAN minimax objective

Consider

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

If the discriminator is perfect, then

$$D(\mathbf{x}) \approx 1 \quad \text{for real data,} \qquad D(G(\mathbf{z})) \approx 0 \quad \text{for fake data.}$$

Then

$$\log D(\mathbf{x}) \approx \log 1 = 0, \qquad \log(1 - D(G(\mathbf{z}))) \approx \log 1 = 0,$$

so the discriminator attains a large value of the objective. The generator, however, wants

$$D(G(\mathbf{z})) \approx 1,$$

which would make

$$\log(1 - D(G(\mathbf{z}))) \to -\infty.$$

Thus the two players have directly opposing goals.

The elegance of this formulation is one of the reasons GANs became so influential. But in practice, the minimax form is not the objective usually used for the generator.

5 Alternate optimization

GAN training is not performed by solving the minimax game exactly. Instead, one alternates between updates of the discriminator and updates of the generator.

5.1 Discriminator step

For fixed generator parameters, update θ_D by maximizing

$$J_D(\theta_D) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))].$$

5.2 Generator step

For fixed discriminator parameters, the naive minimax update would minimize

$$J_G^{\text{minimax}}(\theta_G) = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))].$$

In practice, however, this minimax generator loss often leads to very weak gradients when the discriminator is already good. For this reason, one usually replaces it with the *non-saturating* objective

$$J_G^{\text{NS}}(\theta_G) = -\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log D_{\theta_D}(G_{\theta_G}(\mathbf{z}))].$$

Equivalently, the generator maximizes

$$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log D_{\theta_D}(G_{\theta_G}(\mathbf{z}))].$$

This alternative does not change the intended equilibrium, but it provides stronger gradients during training.

Worked example — Why the non-saturating generator loss helps

The naive generator loss is

$$J_G^{\text{minimax}} = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))].$$

If the discriminator is strong early in training, then for fake samples

$$D(G(\mathbf{z})) \approx 0.$$

In that regime,

$$\log(1 - D(G(\mathbf{z}))) \approx \log 1 = 0,$$

and its gradient with respect to generator parameters can become very small. By contrast, the non-saturating loss

$$J_G^{\text{NS}} = -\mathbb{E}_{\mathbf{z}}[\log D(G(\mathbf{z}))]$$

becomes large when $D(G(\mathbf{z}))$ is close to zero, so it yields a much stronger learning signal. Thus the generator is trained more effectively when its samples are still obviously fake.

6 Game-theoretic view and saddle-point difficulty

GAN training is fundamentally different from standard optimization because it is not minimizing one scalar objective over one set of parameters. It is solving a two-player game:

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

The ideal solution is a saddle point, not a minimum. This makes training intrinsically unstable.

In ordinary optimization, gradient descent attempts to move downhill. In a minimax problem, one player moves downhill while the other moves uphill. As a result:

- iterates may oscillate,
- one player may overpower the other,
- the system may fail to converge even if both losses are being optimized locally.

This is why GAN training historically required many heuristics and architectural tricks.

7 The optimal discriminator and divergence viewpoint

For a fixed generator, one can derive the discriminator that maximizes the original GAN objective pointwise. Let p_G denote the generator distribution induced by G_{θ_G}. Then the optimal discriminator is

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}.$$

Substituting this back into the value function shows that, under idealized assumptions, GAN training corresponds to minimizing a divergence between p_G and p_{data}. In the original GAN, this is closely related to the Jensen–Shannon divergence.

Worked example — Deriving the optimal discriminator

For fixed G, the discriminator maximizes

$$V(D,G) = \int p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x})\, d\mathbf{x} + \int p_G(\mathbf{x}) \log(1 - D(\mathbf{x}))\, d\mathbf{x}.$$

Since this decomposes pointwise in $\mathbf{x}$, for each $\mathbf{x}$ we maximize

$$f(D) = a \log D + b \log(1 - D),$$

where

$$a = p_{\text{data}}(\mathbf{x}), \qquad b = p_G(\mathbf{x}).$$

Differentiate:

$$\frac{df}{dD} = \frac{a}{D} - \frac{b}{1 - D}.$$

Set the derivative to zero:

$$\frac{a}{D} = \frac{b}{1 - D}.$$

Solving for D gives

$$a(1 - D) = bD \quad \Longrightarrow \quad a = (a + b)D \quad \Longrightarrow \quad D^*(\mathbf{x}) = \frac{a}{a + b} = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}.$$

This derivation is elegant, but it also reveals a practical difficulty: if the real and generated distributions have little overlap, the discriminator easily becomes nearly perfect, and useful gradients for the generator may vanish.

8 Mode collapse

One of the most famous failure modes of GANs is *mode collapse*. Instead of learning the full diversity of the data distribution, the generator learns to produce a narrow family of samples that the discriminator happens to judge as plausible.

In the extreme case, many different latent inputs $\mathbf{z}$ are mapped to nearly identical outputs:

$$G(\mathbf{z}_1) \approx G(\mathbf{z}_2) \approx \cdots$$

even when the training distribution contains many distinct modes.

This happens because the generator is rewarded only for fooling the discriminator, not for covering the whole data distribution explicitly. If a small set of outputs is enough to fool the discriminator locally, the generator may settle there.

Mode collapse is therefore not a minor technicality; it is a structural consequence of the adversarial objective and its training dynamics.

9 Deep convolutional GANs (DCGANs)

A major early milestone in GAN development was the Deep Convolutional GAN (DCGAN). The main idea is to instantiate the generator and discriminator with convolutional neural networks suited to images.

9.1 Generator

The generator starts from a latent vector

$$\mathbf{z} \sim p(\mathbf{z}),$$

projects it into a learned feature tensor, and then repeatedly upsamples through transposed convolutions (sometimes called fractionally strided convolutions), typically with batch normalization and nonlinearities.

9.2 Discriminator

The discriminator is a convolutional network that progressively downsamples the image and outputs a scalar real/fake score.

DCGANs were important because they demonstrated that adversarial learning could produce coherent natural-image synthesis and useful latent-space structure with relatively standard deep-learning components.

10 Latent-space structure and interpolation

A striking empirical property of well-trained GANs is that latent space often supports smooth semantic manipulation. If the generator is well behaved, nearby latent vectors produce visually similar outputs, and linear interpolation can produce coherent transitions.

More remarkably, certain vector arithmetic patterns can emerge:

$$\mathbf{z}_G = \mathbf{z}_1 - \mathbf{z}_2 + \mathbf{z}_3.$$

Decoding $\mathbf{z}_G$ can yield a sample combining semantic traits in a meaningful way. This is not guaranteed by theory, but it is one reason adversarially learned latent spaces are so appealing.

Worked example — Latent interpolation in a GAN

Let $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^K$ be two latent vectors. Define

$$\mathbf{z}(\alpha) = (1-\alpha)\mathbf{z}_1 + \alpha\mathbf{z}_2, \qquad \alpha \in [0,1].$$

The generated path is

$$G(\mathbf{z}(0)),\ G(\mathbf{z}(0.1)),\ \ldots,\ G(\mathbf{z}(1)).$$

If the latent space is well organized, these outputs often vary smoothly in appearance. This suggests that the generator has learned a continuous mapping from latent space into a structured manifold of realistic samples.

11 Why the original GAN loss can be problematic

The original GAN objective is elegant, but its gradients can be poor when the support of p_G and p_{data} are far apart. In that case:

- the discriminator becomes nearly perfect,
- the Jensen–Shannon-type objective saturates,
- the generator receives weak or unstable gradients.

This motivates the search for alternative objectives that behave more smoothly even when generator samples are still far from the data distribution.

12 Wasserstein GANs

A major advance was to replace the original adversarial objective with one based on the Wasserstein-1 distance, also called Earth Mover's Distance (EMD).

Intuitively, the Wasserstein distance measures how much "mass" must be moved, and how far, to transform one distribution into another. Unlike the Jensen–Shannon divergence, it remains meaningful even when distributions have disjoint support.

The Wasserstein GAN objective is based on the Kantorovich–Rubinstein dual formulation:

$$W(p_{\text{data}}, p_G) = \sup_{\|D\|_L \le 1} \left(\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[D(G(\mathbf{z}))] \right),$$

where the supremum is over all 1-Lipschitz functions D.

In this setting the discriminator becomes a *critic*: it no longer outputs a probability of being real, but an unrestricted scalar score. The generator minimizes the estimated Wasserstein distance.

Worked example — Why Wasserstein distance gives gradients when GAN loss saturates

Suppose the real and fake distributions are concentrated around far-apart regions. In a classical GAN, the discriminator can separate them almost perfectly, which drives the sigmoid outputs to saturation and makes generator gradients small. In a Wasserstein GAN, the critic instead learns a scalar function whose difference in expectation

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[D(G(\mathbf{z}))]$$

approximates transport cost. Even when the distributions are far apart, this difference remains informative and varies approximately linearly with their displacement. Therefore the generator can receive useful gradients over a wider range of training conditions.

13 The Lipschitz constraint

The dual Wasserstein formulation is valid only if the critic is 1-Lipschitz. This means

$$|D(\mathbf{x}) - D(\mathbf{x}')| \le \|\mathbf{x} - \mathbf{x}'\| \qquad \text{for all } \mathbf{x}, \mathbf{x}'.$$

Enforcing this constraint is the main practical challenge in Wasserstein GANs.

The earliest WGAN implementation used weight clipping: constrain each critic weight to lie in a small interval. This is easy to implement but often harms optimization. Later methods replaced weight clipping with more principled techniques such as gradient penalty, which are now standard in practice.

14 Progressive growing for high-resolution generation

Generating high-resolution images is much harder than generating small ones. The output space is larger, details matter more, and training instability becomes worse. A successful strategy is *progressive growing*.

The idea is to train the GAN first on very low-resolution images, for example 4×4, and then gradually add layers to both generator and discriminator so that resolution increases over time:

$$4 \times 4 \to 8 \times 8 \to 16 \times 16 \to \cdots$$

During each transition, the new layers are smoothly faded in rather than switched on abruptly.

The intuition is that the model first learns coarse global structure and only later learns fine detail. This significantly improves training stability and sample quality at high resolution.

15 Conditional GANs

A conditional GAN augments both generator and discriminator with side information $\mathbf{y}$. Instead of learning an unconditional sampler

$$\mathbf{z} \mapsto G(\mathbf{z}),$$

the generator learns

$$G(\mathbf{y}, \mathbf{z}) \mapsto \tilde{\mathbf{x}}.$$

The discriminator also receives the condition, so it judges whether a sample is both realistic and compatible with $\mathbf{y}$.

This changes the target from modeling

$$p(\mathbf{x})$$

to modeling

$$p(\mathbf{x} \mid \mathbf{y}).$$

Conditional GANs are especially important because they enable controlled generation:

- class-conditioned image synthesis,
- age progression,
- text-to-image generation,
- image-to-image translation.

Worked example — Conditional GAN objective

Let $\mathbf{y}$ be a condition variable. Then the discriminator objective becomes

$$J_D = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}}[\log D(\mathbf{x}, \mathbf{y})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}),\, \mathbf{y}\sim p_{\text{data}}}[\log(1 - D(G(\mathbf{y}, \mathbf{z}), \mathbf{y}))].$$

The generator is trained to make

$$D(G(\mathbf{y}, \mathbf{z}), \mathbf{y})$$

large, meaning that generated samples should be both realistic and condition-consistent.

16 Image-to-image translation

A particularly important case of conditional generation is image-to-image translation. Here the condition is itself an image:

$$G(\mathbf{x}, \mathbf{z}) = \mathbf{y}.$$

Examples include:

- semantic labels $\to$ photo,
- sketch $\to$ photo,
- grayscale $\to$ color,
- day $\to$ night.

Often the generator is implemented as an encoder–decoder or U-Net architecture, while the discriminator evaluates whether the output image looks realistic given the input image.

This setting shows how GANs can be used not only to sample from noise, but also to learn structured mappings between domains.

17 Cycle-consistent adversarial learning

Sometimes paired input-output examples are unavailable. For example, one may have a set of horse images and a set of zebra images, but no aligned horse/zebra pairs. Cycle-consistent adversarial learning addresses this.

Let

$$G_{A\to B} : A \to B, \qquad G_{B\to A} : B \to A$$

be two generators, each with its own discriminator. Adversarial losses ensure that translated samples look realistic in the target domain. But realism alone is not enough: many mappings from A to B could fool a discriminator. So one adds a *cycle consistency* loss:

$$G_{B\to A}(G_{A\to B}(\mathbf{x}_A)) \approx \mathbf{x}_A,$$

and similarly in the opposite direction.

A standard cycle-consistency penalty is

$$L_{\text{cyc}} = \mathbb{E}_{\mathbf{x}_A}[\|G_{B\to A}(G_{A\to B}(\mathbf{x}_A)) - \mathbf{x}_A\|_1] + \mathbb{E}_{\mathbf{x}_B}[\|G_{A\to B}(G_{B\to A}(\mathbf{x}_B)) - \mathbf{x}_B\|_1]\,.$$

Worked example — Why cycle consistency helps without paired data

Suppose we want to translate from domain A to domain B using only unpaired samples. An adversarial loss alone ensures that

$$G_{A\to B}(\mathbf{x}_A)$$

looks like a sample from domain B. But this does not guarantee that it preserves the content of $\mathbf{x}_A$. The generator could map many different A-samples to the same plausible B-sample. By adding the reconstruction requirement

$$G_{B\to A}(G_{A\to B}(\mathbf{x}_A)) \approx \mathbf{x}_A,$$

we constrain the mapping to remain approximately invertible on the data manifold. This greatly reduces the space of degenerate solutions.

18 Adversarial autoencoders

Adversarial learning can also be combined with autoencoders. An *adversarial autoencoder* (AAE) keeps the reconstruction structure of an autoencoder but replaces KL-based latent regularization with an adversarial game in latent space.

Let the encoder produce a latent code

$$\mathbf{z} = f_\theta(\mathbf{x}),$$

and let the decoder reconstruct

$$\tilde{\mathbf{x}} = g_\theta(\mathbf{z}).$$

The training has two phases:

1. **Reconstruction phase:** update encoder and decoder to minimize reconstruction loss.
2. **Regularization phase:** use a discriminator in latent space to distinguish prior samples

   $$\mathbf{z} \sim p(\mathbf{z})$$

   from encoded samples

   $$\mathbf{z} = f_\theta(\mathbf{x}),$$

   and update the encoder to fool this discriminator.

The effect is to match the aggregated latent distribution to a chosen prior without requiring a closed-form KL divergence.

This is useful when the desired prior is complicated or empirical.