Artificial Intelligence III: Advanced Deep Learning Methods

Ana Neacșu & Vlad Vasilescu

Chapter 3: Generative Adversarial Networks

National University of Science and Technology POLITEHNICA Bucharest, Romania
BIOSINF Master Program

October 2024

Introduction

What is a Generative Adversarial Network (GAN)?

- A Generative Adversarial Network, or GAN, is a class of machine learning frameworks used for generative modeling.
- It consists of two neural networks:
  - **Generator** (G): Creates fake data intended to mimic real data.
  - **Discriminator** (D): Evaluates the authenticity of the data, distinguishing between real and generated data.
- Both networks are trained simultaneously in a game-theoretic manner:
  - The generator attempts to produce data that fools the discriminator.
  - The discriminator strives to correctly identify real data from generated data.

Generative Adversarial Networks

Figure: "Two imaginary celebrities that were dreamed up by a random number generator. "

Source : https://research.nvidia.com/publication/2017-10_Progressive-Growing-of

Why should we care about GANs?

- GANs are achieving state-of-the-art results in a large variety of image generation tasks.
- There's been a veritable explosion in GAN publications over the last few years.
- GANs are stimulating new theoretical interest in min-max optimization problems and "smooth games".
- There is a nice connection with the Adversarial AI tehory that we will study in this semester.

Why should we care about GANs? II

- We have mainly seen **discriminative models** so far
  - Given in input X, we predict a label y.
  - The task is to estimate $P(y|x)$.
- Discriminative models have several **key limitations**:
  - Can't model $P(X)$, i.e. the probability of seeing a certain image.
  - Thus, can't sample from $P(X)$, i.e. can't generate new images.
- Generative models (in general) **cope with all of above**:
  - Are able to model $P(X)$.
  - Can generate new images.

Why care about GANs: Hyper-realistic Image Generation

Figure: StyleGAN: image generation with hierarchical style transfer.

https://arxiv.org/abs/1812.04948

Why care about GANs: Conditionally Generative Models

Conditional GANs: High-resolution image synthesis via semantic labeling

Figure: Input: Segmentation

Figure: Output: Synthesized Image

---

https://research.nvidia.com/publication/2017-12_High-Resolution-Image-Synthesis

Why care about GANs: Image Super Resolution

SRGAN: One of the best photo-realistic super resolution systems

Figure: Bicubic Interp.

Figure: SRGAN

Figure: Original Image

---

https://arxiv.org/abs/1609.04802

Generative Models

Generative Modeling

Generative Models estimate the probabilistic process that generated a set of observations $\mathcal{D}$.

- $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^n$: supervised generative models learn the joint distribution $p(\mathbf{x}^i, \mathbf{y}^i)$, often to compute $p(\mathbf{y}^i|\mathbf{x}^i)$.

- $\mathcal{D} = \{\mathbf{x}^i\}_{i=1}^n$: unsupervised generative models learn the distribution of $\mathcal{D}$ for clustering, sampling, etc. We can:
  - directly estimate $p(\mathbf{x}^i)$
  - introducing **latents** $\mathbf{y}^i$ and estimate $p(\mathbf{y}^i|\mathbf{x}^i)$.

Generative Modeling: Unsupervised Parametric Approaches

- **Direct Estimation**: Choose a parameterized family $p(\mathbf{x}|\theta)$ and learn θ by maximizing the log-likelihood

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log\, p(\mathbf{x}^i|\theta).$$

- **Latent Variable Models**: Define a joint distribution$p(\mathbf{x}, \mathbf{y}|\theta)$ and learn θ by maximizing the log-marginal likelihood

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log \int_{\mathbf{z}^i} p(\mathbf{x}^i, \mathbf{z}^i|\theta)\, d\mathbf{z}.$$

Both approaches require that $p(x|\theta)$ is easy to evaluate.

Generative Modeling: Models for (Very) Complex Data

Figure: How can we learn such models for very complex data?

Generative Modeling: Normalizing Flows and VAEs

Solution : Design parameterized densities with huge capacity!

- **Normalizing flows**: sequence of non-linear transformations to a simple distribution $p_z(\mathbf{z})$

$$p(\mathbf{x}|\theta_{0:k}) = p_z(\mathbf{z}) \text{ where } \mathbf{z} = f_{\theta_k}^{-1} \circ \cdots \circ f_{\theta_1}^{-1} \circ f_{\theta_0}^{-1}(\mathbf{x}).$$

  $f_{\theta_i}^{-1}$ must be invertible with tractable log-det.

- **VAEs**: latent-variable models where inference networks specify parameters

$$p(\mathbf{x}, \mathbf{y}|\theta) = p(\mathbf{x}|f_\theta(\mathbf{y}))p_y(\mathbf{y}).$$

  The marginal likelihood is maximized via the ELBO.

Training GANs

GANs: Density-Free Models

Generative Adversarial Networks (GANs) instead use an unrestricted generator $G_{\theta_g}(\mathbf{z})$ such that

$$p(\mathbf{x}|\theta_g) = p_z(\{\mathbf{z}\}) \text{ where } \{\mathbf{z}\} = G_{\theta_g}^{-1}(\mathbf{x}).$$

Main problems :

- the inverse image of $G_{\theta_g}(\mathbf{z})$ may be huge!
- it's likely intractable to preserve volume through $G(\mathbf{z}; \theta_g)$.

So, we can't evaluate $p(\mathbf{x}|\theta_g)$ and we can't learn θ_g by maximum likelihood.

GANs: Discriminators

GANs learn by comparing model samples with examples from $\mathcal{D}$.

- Sampling from the generator is easy:

$$\hat{\mathbf{x}} = G_{\theta_g}(\hat{\mathbf{z}}), \text{ where } \hat{\mathbf{z}} \sim p_z(\mathbf{z}).$$

- Given a sample $\hat{\mathbf{x}}$, a discriminator tries to distinguish it from true examples:

$$D(\mathbf{x}) = \text{Pr } (\mathbf{x} \sim p_{data}).$$

- The discriminator "supervises" the generator network.

GANs: Generator + Descriminator

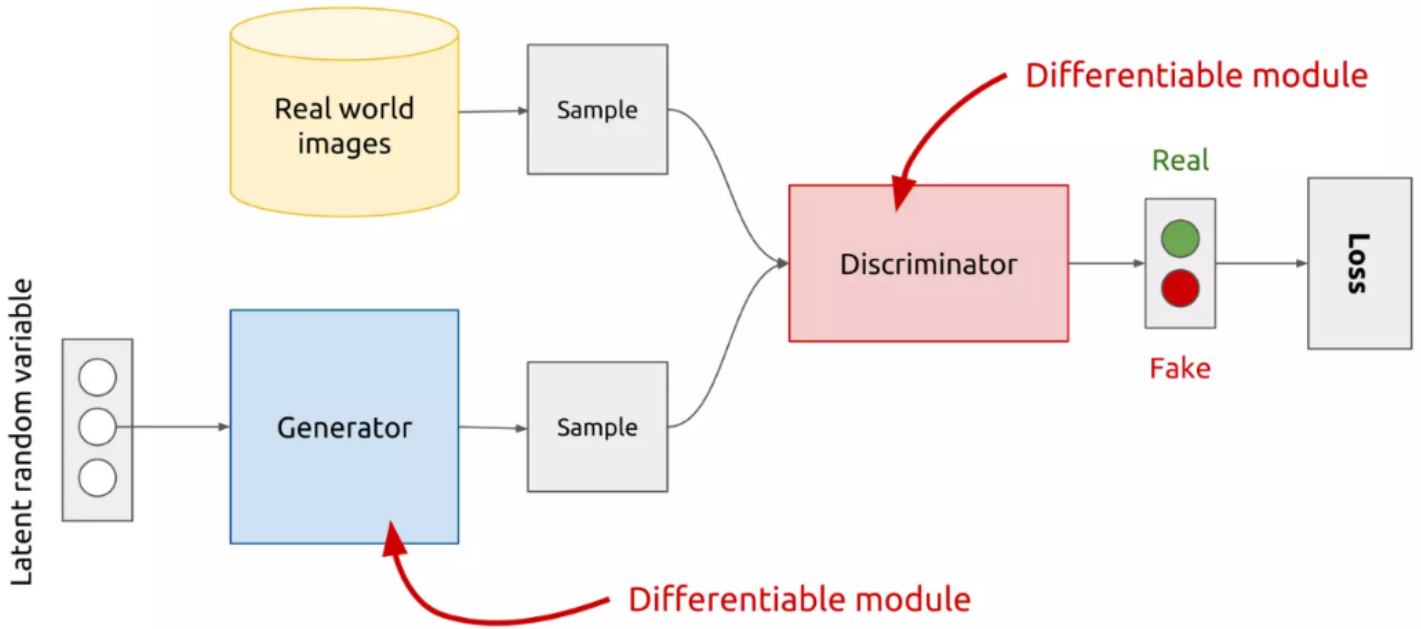


Figure: The generator maps a latent random variable into a sample, while the discriminator tries to distinguish between real and fake samples, providing feedback to the generator to improve its performance.

GANs: Goodfellow et al. (2014)

- Let $\mathbf{z} \in \mathbb{R}^m$ and $p_z(\mathbf{z})$ be a simple base distribution.
- The generator $G_{\theta_g}(\mathbf{z}) : \mathbb{R}^m \rightarrow \tilde{\mathcal{D}}$ is a deep neural network.
  - $\tilde{\mathcal{D}}$: is the manifold of generated examples.
- The discriminator $D_{\theta_d}(\mathbf{x}) : \mathcal{D} \cup \tilde{\mathcal{D}} \rightarrow (0, 1)$ is also a deep neural network.

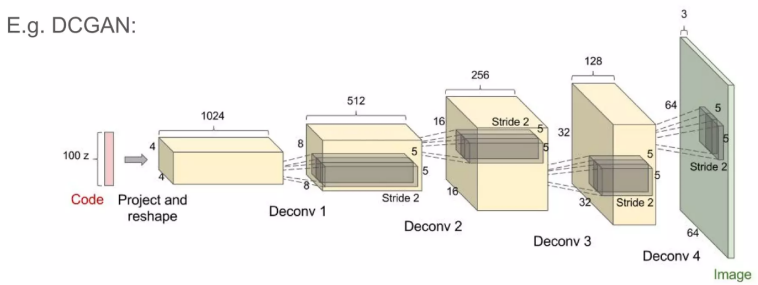


Figure: Generator

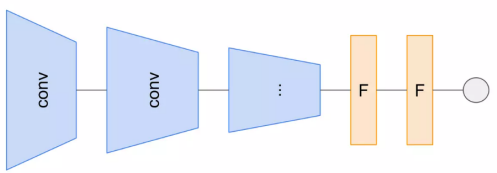


Figure: Discriminator

---

https://arxiv.org/abs/1511.06434

GANs: Saddle-Point Optimization

Saddle-Point Optimization: learn $G_{\theta_g}(\mathbf{z})$ and $D_{\theta_d}(\mathbf{x})$ jointly via the objective $V(\theta_d, \theta_g)$:

$$\min_{\theta_g}\max_{\theta_d}\mathbb{E}_{p_{\text{data}}}[\log\ D_{\theta_d}(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log\ (1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z})))].$$

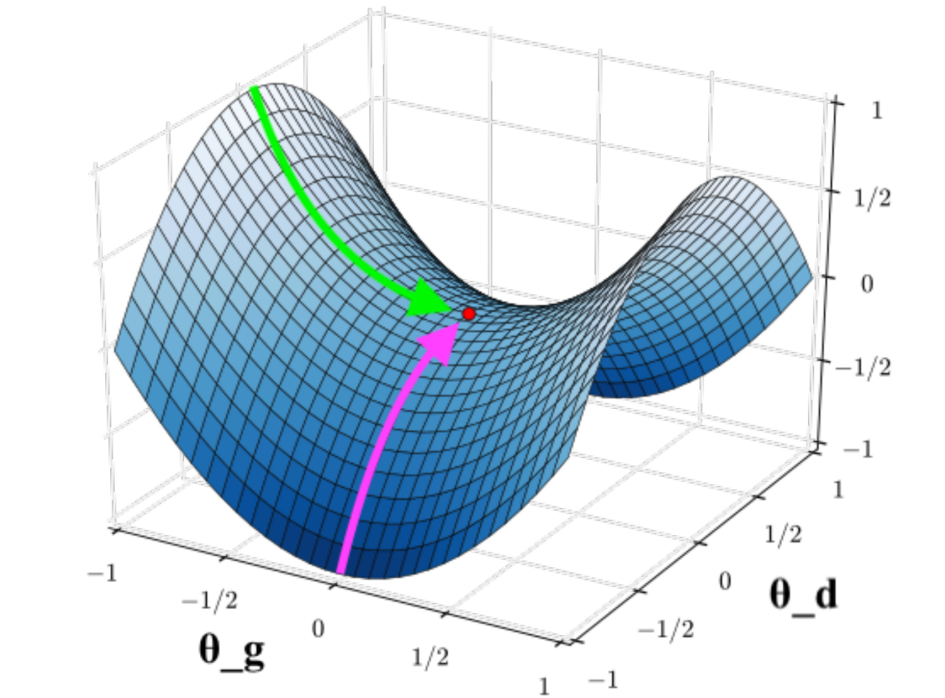


Figure: The minimax objective is represented as a saddle point in this plot. The goal is to find a configuration where the generator's output (represented by θ_g) is indistinguishable from real data (represented by θ_d).

GANs: Optimal Discriminators

Claim: Given G_{θ_g} defining an implicit distribution $p_g = p(\mathbf{x}|\theta_g)$, the optimal descriminator is

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

Proof Sketch:

$$V(\theta_d, \theta_g) = \int_{\mathcal{D}} p_{data}(\mathbf{x}) \log\, D(\mathbf{x})\, d\mathbf{x} + \int_{\tilde{\mathcal{D}}} p(\mathbf{z}) \log(1 - D(G_{\theta_g}(\mathbf{z})))\, d\mathbf{z}$$

$$= \int_{\mathcal{D} \cup \tilde{\mathcal{D}}} p_{data}(\mathbf{x}) \log\, D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))\, d\mathbf{x}$$

Maximizing the integrand for all $\mathbf{x}$ is sufficient and gives the result.

GANs: Jensen-Shannon Divergence and Optimal Generators

Given an optimal discriminator $D^*(\mathbf{x})$, the generator objective is

$$\begin{aligned} C(\theta_g) &= \mathbb{E}_{p_{data}}\left[\log\ D^*_{\theta_d}(\mathbf{x})\right] + \mathbb{E}_{p_g(\mathbf{x})}\left[\log\ \left(1 - D^*_{\theta_d}(\mathbf{x})\right)\right] \\ &= \mathbb{E}_{p_{data}}\left[\log\ \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}\right] + \mathbb{E}_{p_g(\mathbf{x})}\left[\log\ \left(1 - \frac{p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}\right)\right]. \end{aligned}$$

$$\alpha\frac{1}{2}\mathsf{KL}\left(p_{data}\middle|\frac{p_{data}+p_g}{2}\right) + \frac{1}{2}\mathsf{KL}\left(p_g\middle|\frac{p_{data}+p_g}{2}\right)$$

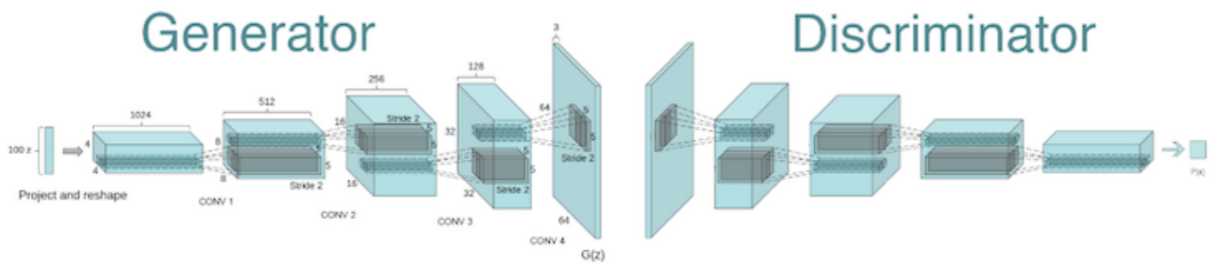
Nash equilibrium

$C(\theta_g)$ achieves its global minimum at $p_g = p_{data}$ given an optimal discriminator!

GANs: Learning Generators and Discriminators

Putting these results to use in practice:

- **High-capacity discriminators** D_{θ_d}: approximate the Jensen-Shannon divergence when close to global maximum.
- D_{θ_d} is a "differentiable module", a binary calssificator.
- **We can use** D_{θ_d} **to learn** G_{θ_g} with our favourite gradient descent method.



---

https://arxiv.org/abs/1511.06434

GANs: Training Procedure

Algorithm 1: Training GANs steps

Input: N: number of iterations, K: number of inner iterations, m: number of samples, α_d: learning rate for discriminator, α_g: learning rate for generator

for $i = 1$ **to** N **do**

  for $k = 1$ **to** K **do**

    •Sample noise samples $\{ z^2, \ldots, z^m \} \sim p_z(z)$;

    •Sample examples $\{x^1, \ldots, x^m\}$ from $p_{\text{data}}(x)$;

    •Update the discriminator D_{θ_d}:

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

  •Sample noise samples $\{ z^1, \ldots, z^m \} \sim p_z(z)$;

  •Update the generator G_{θ_g}:

$$\theta_g = \theta_g - \alpha_g \nabla_{\theta_g} \frac{1}{m} \sum_{j=1}^{m} \log(1 - D(G(z^j)))$$

Good Practices for Training GANs

1. Learning Rates

- Use separate learning rates: α_d (discriminator) $> \alpha_g$ (generator).
- Typical values: $\alpha_d \in (0.0002, 0.001)$; $\alpha_g \approx 0.0001$.
- Monitor loss; adjust learning rates if needed.

2. Batch Size

- Experiment with batch sizes (32 to 256).
- Smaller sizes introduce noise; larger sizes stabilize training.

Update Iterations and Architecture

3. Update Iterations

- Balance updates between generator and discriminator.
- Common practice: K (discriminator updates) from 1 to 5 per generator update.
- Adjust K dynamically based on model performance.

4. Model Architecture

- Experiment with layers and units in both networks.
- Use ReLU or Leaky ReLU for hidden layers; Sigmoid or Tanh for outputs.

Regularization and Monitoring

5. Regularization Techniques

- Use dropout or batch normalization to improve stability.
- Consider methods like WGAN with gradient penalty for stabilization.

6. Monitoring and Evaluation

- Visualize generated samples regularly.
- Track losses closely to maintain balance between networks.

Conclusion: Careful parameter management and update balancing are key to effective GAN training.

Main problems

Problems with GANs

- **Vanishing gradients**: the discriminator becomes "too good" and the generator gradient vanishes.
- **Non-Convergence**: the generator and discriminator oscillate without reaching an equilibrium.
- **Mode Collapse**: the generator distribution collapses to a small set of examples.
- **Mode Dropping**: the generator distribution doesn't fully cover the data distribution.

Problems: Vanishing Gradients

- The **minimax** objective saturates when D_{θ_d} is close to perfect:

$$V(\theta_d, \theta_g) = \mathbb{E}_{p_{data}}[\log\ D_{\theta_d}(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log\ (1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z})))].$$

- A **non-saturating heuristic** objective for the generator is

$$J(G_{\theta_g}) = -\mathbb{E}_{p_z(\mathbf{z})}[\log\ (D_{\theta_d}(G_{\theta_g}(\mathbf{z})))].$$

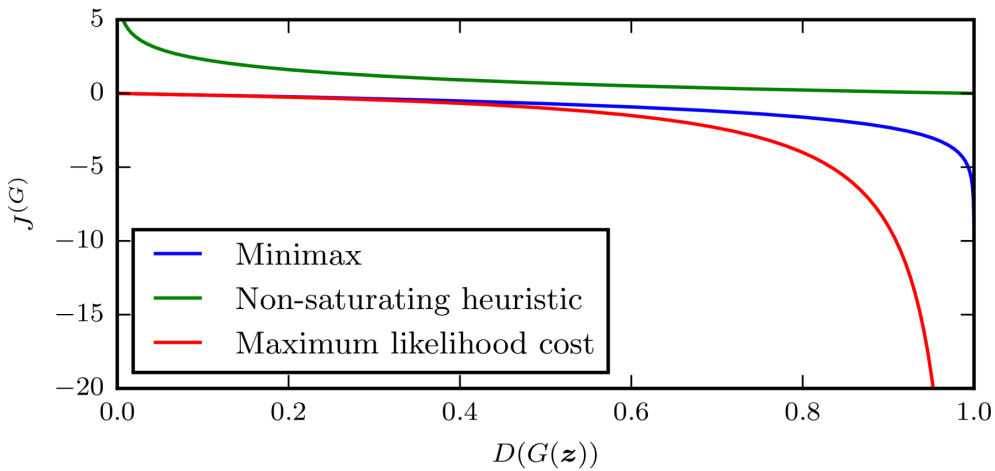


Figure: The minimax objective (blue), non-saturating heuristic (green), and maximum likelihood cost (red) are shown in this plot. The non-saturating heuristic provides a more stable gradient than the minimax objective, particularly when the discriminator becomes strong.

https://arxiv.org/abs/1701.00160

Addressing Vanishing Gradients

Possible Solutions:

- **Change Objectives:** use the non-saturating heuristic objective, maximum-likelihood cost, etc.
- **Limit Discriminator:** restrict the capacity of the discriminator.
- **Schedule Learning:** try to balance training D_{θ_d} and G_{θ_g}.

Problems: Non-Convergence

The problem: Simultaneous gradient descent is not guaranteed to converge for minimax objectives.

- Goodfellow et al. only showed convergence when updates are made in the function space.
- The parameterization of D_{θ_d} and G_{θ_g} results in highly non-convex objective.
- In practice, training tends to oscillate – updates "undo" each other.

Problems: Addressing Non-Convergence

Possible Solutions: Lots of hacks proposed lately! Check ganhacks

- **Use a spherical Z**
  - Dont sample from a Uniform distribution.
  - Sample from a gaussian distribution
  - When doing interpolations, do the interpolation via a great circle, rather than a straight line from point A to point B.
- **Use BatchNorm:**
  - Construct different mini-batches for real and fake, i.e. each mini-batch needs to contain only all real images or all generated images.
  - When BatchNorm is not an option use instance normalization (for each sample, subtract mean and divide by standard deviation).
- **Avoid Sparse Gradients: ReLU, MaxPool**
  - The stability of the GAN game suffers if you have sparse gradients
  - LeakyReLU $=$ good (in both G and D)
  - For Downsampling, use: Average Pooling, Conv2d $+$ stride
  - For Upsampling, use: PixelShuffle, ConvTranspose2d $+$ stride
- **Track failures early**:
  - D loss goes to 0: failure mode
  - When things are working, D loss has low variance and goes down over time vs having huge variance and spiking

Problems: Mode Collapse and Mode Dropping

The problem: SGD may optimize the max-min objective

$$\max_{\theta_d}\min_{\theta_g}\mathbb{E}_{p_{data}}[\log\ D_{\theta_d}(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log\ (1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z})))].$$

Intuition: the generator maps all z values to the x that is mostly likely to fool the discriminator.

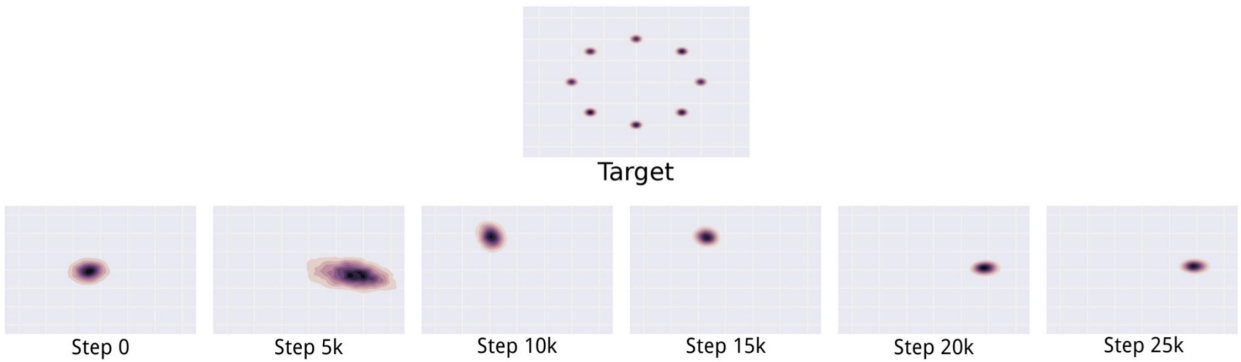


Figure: This figure shows a potential problem with mode collapse, where the generator repeatedly generates similar samples, leading to a narrow distribution that fails to capture the diversity of the data.

---

https://arxiv.org/abs/1701.00160

A Possible Solution: Alternative Divergences

There are a large variety of divergence measures for distributions:

- f-Divergences: (e.g. Jensen-Shannon, Kullback-Leibler)

$$D_f(P||Q) = \int_{\mathcal{X}} q(\mathbf{x}) f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) d\mathbf{x}$$

  - f-GANs

- **Integral Probability Metrics**: (e.g. Earth Movers Distance, Maximum Mean Discrepancy)

$$\gamma_F(P||Q) = \sup_{f \in F} \left| \int_{\mathcal{X}} f dP - \int_{\mathcal{X}} f dQ \right|$$

  - Wasserstein GANs, Fisher GANs, Sobolev GANs and more.

A Possible Solution: Wasserstein GANs

Wasserstein GANs: Strong theory and excellent empirical results.

- "In no experiment did we see evidence of mode collapse for the WGAN algorithm."

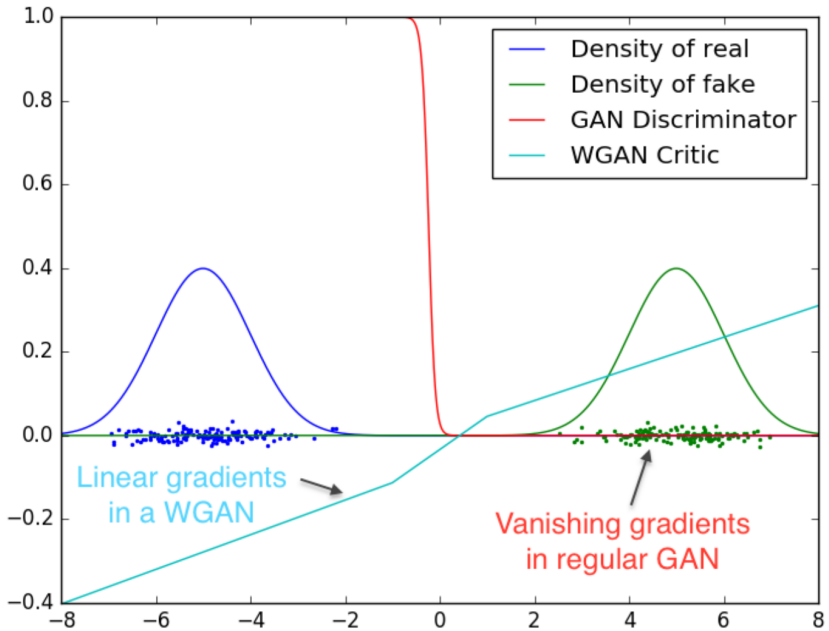
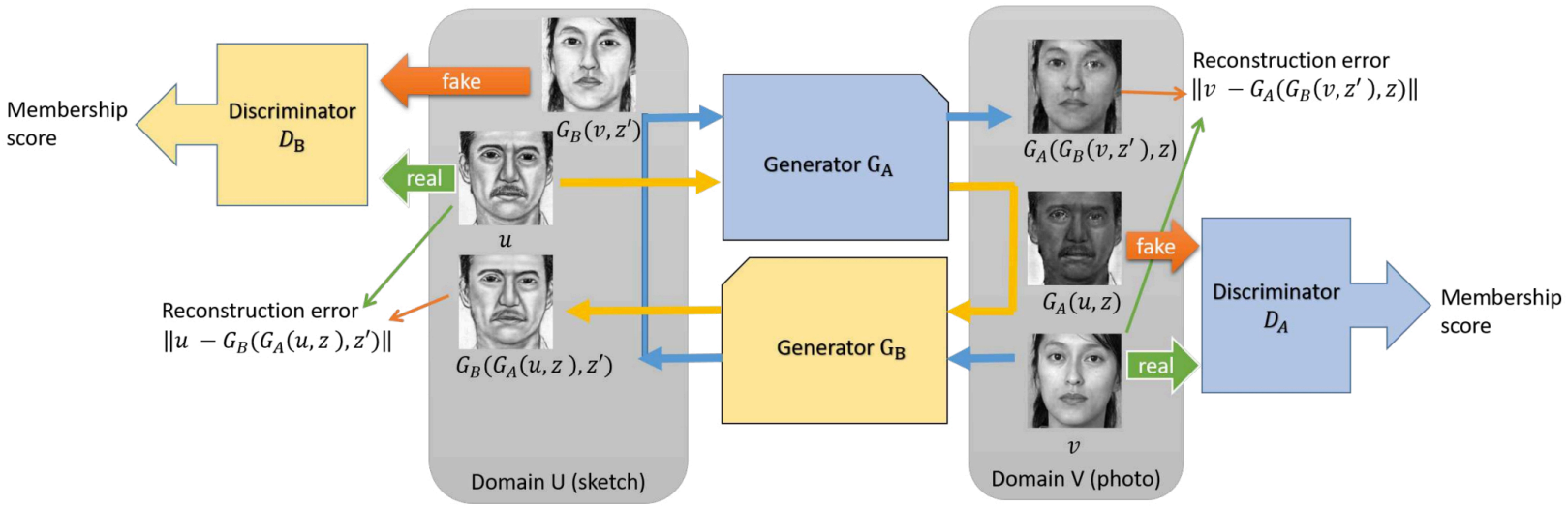


Figure: The plot shows the distributions of real and fake data (blue and green), as well as the discriminator's output (red). This plot highlights the advantages of Wasserstein GANs over standard GANs, namely the ability to provide smoother gradients and mitigate the problem of vanishing gradients.

https://arxiv.org/abs/1701.07875

GAN Advanced Methods

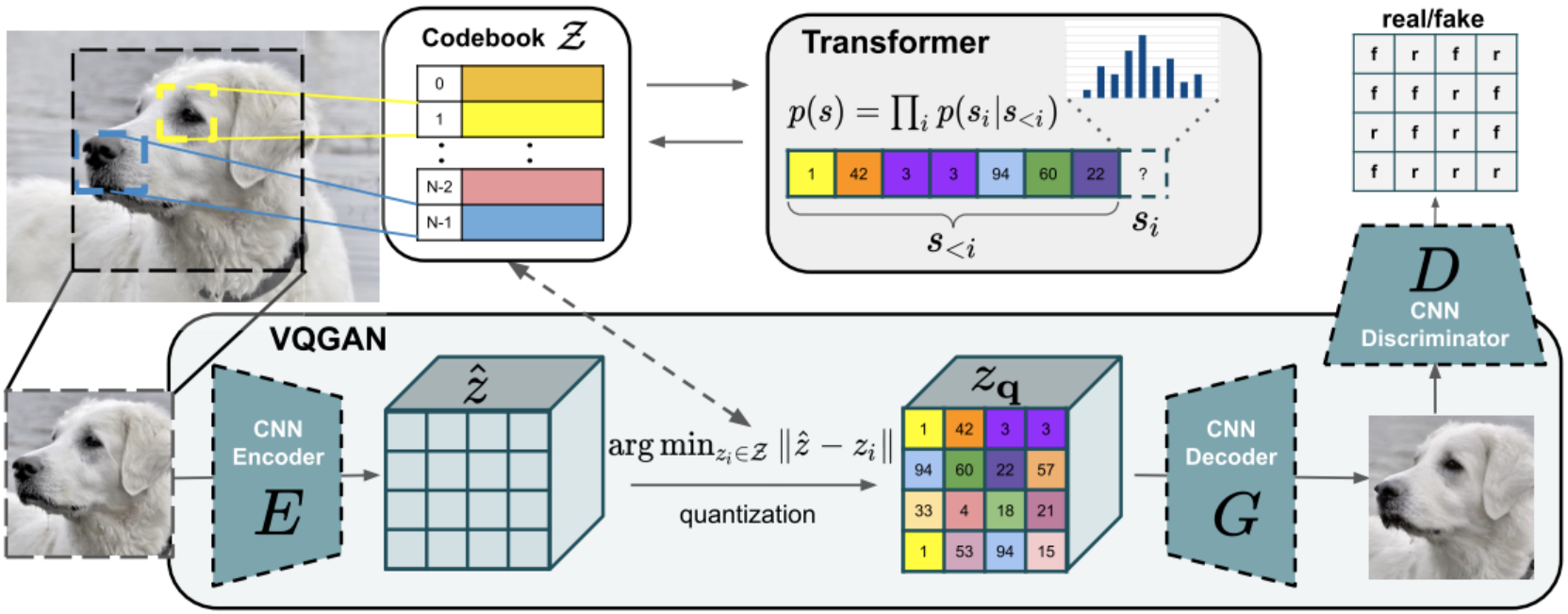
DualGAN: Image-to-Image Translation



Architecture of DualGAN. Two different GANs are trained, (G_A, D_A) learns the primal task, while (G_B, D_B) learns the inverse task. [Source]

- We want to learn $G_A : U \to V$ (primal task) and $G_B : V \to U$ (dual task)
- Both GANs are trained simoultaneously
- Doesn't require training on a dataset of corresponding input-output images

Vector Quantised GAN (VQGAN)

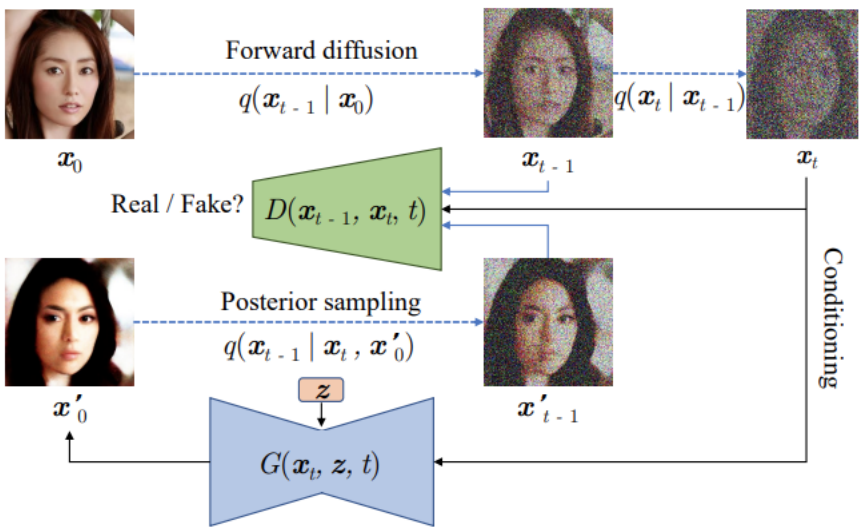


Architecture and workflow of VQGAN. [Source]

- Each encoded image region is mapped onto a discrete *codebook* $\mathcal{Z}$
- An autoregressive Transformer models the sequence of codebooks, learning the relationships between "different regions"
- The Transformer generates sequences of codes s that are mapped by G to new images

Denoising Diffusion GANs

Tackling the generative learning trilemma with denoising diffusion gans



Training of diffusion GAN

- GAN framework incorporated into a diffusion process
- Generating a new image corresponds to running several diffusion steps, where each step is conditioned on the previous output
- Adding the discriminator during training helps in offering better stability and reducing the possibility of mode collapse

Conclusions

Summary

Recap:

- GANs are a class of density-free generative models with (mostly) unrestricted generator functions.
- Introducing adversial discriminator networks allows GANs to learn by minimizing the Jensen-Shannon divergence.
- Concurrently learning the generator and discriminator is challenging due to:
  - Vanishing Gradients,
  - Non-convergence due to oscillation
  - Mode collapse and mode dropping.
- A variety of alternative objective functions are being proposed.

Agknowledgements and References

There are lots of excellent references on GANs:

- Sebastian Nowozin's presentation at MLSS 2018.
- NIPS 2016 tutorial on GANs by Ian Goodfellow.
- A nice explanation of Wasserstein GANs by Alex Irpan.
- A survey on GANs and their applications by J. Y. Zhang et al. (2021).
- A comprehensive overview of GANs by K. Fréchet et al. (2019).
- Generative Adversarial Networks: A Survey and Taxonomy by M. Odena et al. (2022) that reviews recent advancements in GAN frameworks.
- Generative Adversarial Networks for Data Augmentation by T. Saito et al. (2022) showcasing applications of GANs in data augmentation.

Bonus: Optimal Discriminators Cont.

The integrand

$$h(D(\mathbf{x})) = p_{data}(\mathbf{x})\log\ D(\mathbf{x}) + p_g(\mathbf{x})\log(1 - D(\mathbf{x}))$$

is concave for $D(\mathbf{x}) \in (0, 1)$. We take the derivative and compute a stationary point in the domain:

$$\frac{dh(D(\mathbf{x}))}{dD(\mathbf{x})} = \frac{p_{data}(\mathbf{x})}{D(\mathbf{x})} - \frac{p_g(\mathbf{x})}{1 - D(\mathbf{x})} = 0$$

$$\implies D(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}.$$

This minimizes the integrand over the domain of the discriminator, completing the proof.