16/1/2026

AI Factory

Generative and Self-Supervised ML

Implicit Models

Javier Béjar - UPC

BSC AI Factory Partners

Affiliated entities

Introduction

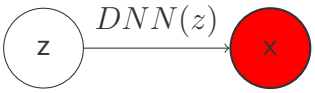
Generative models so far

- All the generative models that we have seen are based on likelihood
  - Exact likelihood: Autoregressive and flow models
  - Approximate likelihood (ELBO): Variational AutoEncoders
- All these models allow to:
  - Train to estimate a probability distribution
  - Compute likelihood of new samples
  - Obtain a latent representation
  - Obtain new samples

Just Samples!

- There are applications where we only need samples, why not obtain a model that only cares about that?
- We want models that
  - Generate samples not only from the training dataset (variations)
  - Able to interpolate samples between those seen on training
  - Obtain samples aware of the underlying factors of the process that generates the data
  - For instance: Different viewing angles, changes of illumination, object features (shape, colour, elements composing the object)

- Implicit models will be able to generate samples z from a reference distribution
- Samples from z will pass through a DNN to compute the samples



- The DNN will not be trained to estimate explicitly a probability density function
- Given a sampler $x = q_\phi(z) = DNN(z;\phi)$ where $z \sim p(z)$
- There is no explicit distribution for p_{data} or p_{model}, only samples
- The goal is to make p_{data} and p_{model} as close as possible

- Our problem now is that we do not have a simple way of comparing both distributions
- Before: Explicit distribution means to be able to measure likelihood and compare distributions (for instance using KL-Divergence), that gives an objective function to optimize
- Now: Implicit distribution means no likelihood, we can only obtain samples, that means to use other similarity measures respect to the samples that will have a different behaviour than likelihood (no theory to back up decisions)

Generative Adversarial Networks

- Generative Adversarial Networks (GANs) were defined in the paper Generative Adversarial Nets (Goodfellow et al., 2014)
- The idea is to work with two models:
  - The Generator (G) captures the data distribution and is able to generate samples
  - The Discriminator (D) estimates the probability of a sample of being either from the original data or from G
- The training is framed as an adversarial min-max game
  - The Generator tries to maximize the probability of their samples
  - The Discriminator tries to minimize that probability

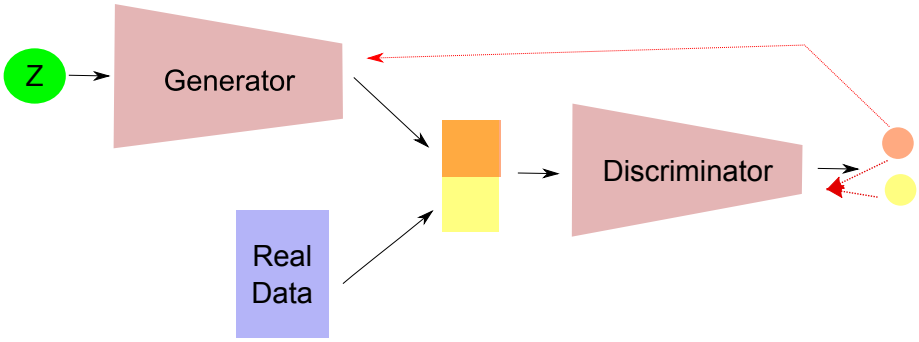
GAN Loss

- The discriminator D will try to maximize the log likelihood for a binary classification problem (real = 1, fake=0)
- $x \sim p_{data}$ are real data samples so $\mathbb{E}_{x \sim p_{data}}[\log D(x)]$ should be increased
- $z \sim p(z)$ is used to generate fake examples using $G(z)$, the discriminator should increase $\mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$
- The goal of the minimax game can be defined as:

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

- If equilibrium is reached then the discriminator should not be able to tell which samples are fake

GAN training schema



Evaluation of GANs

How can we evaluate GANs results?

- The goal is not only to fool the discriminator, but to obtain realistic samples
- We could do it by just generating a few samples that look real, and ignoring most of the probability space
  - This corresponds to a problem observed training GANs called mode colapse
- We should evaluate in some way how diverse are our samples and how well they cover the range of possibilities
- There is not a simple way to do this, specially in high dimensionality
- In some domains we could resort to human evaluation, but it is not an objective measure

Inception Score

- In the image domain, we could use as reference an image classifier for measuring how recognizable and diverse is the content of the images generated
- The inception score uses an InceptionV3 network trained on the ImageNet dataset (1000 classes)
- The classification network will return for an image a distribution for the possible labels
- If the images are good enough, the network will give a high probability to one of the classes, if not, the probability will be distributed along many classes
- If the images are not diverse, the distribution of the predictions will have a low entropy

- Inception Score is not enough for evaluating image diversity (generating one image for each class gives a perfect score)
- The Fréchet Inception Distance uses the features computed by the third pooling layer of the InceptionV3 architecture (2048 features)
- The distance is computed from the mean and covariance of these features for the generated images and the images used to train the GAN

$$d^2((\mu, \Sigma), (\mu_w, \Sigma_w)) = ||\mu - \mu_w||_2^2 + tr(\Sigma + \Sigma_w - 2(\Sigma\Sigma_w)^{\frac{1}{2}})$$

- Many generated images are needed for a faithful evaluation (in the range of tens of thousands)

GANs problems

Mode Collapse

- Mode collapse refers to a situation where the discriminator only generates a specific subset of the data
- The original GAN loss function optimizes the Jensen-Shannon Divergence among the distribution of the generator and the distribution of the real data
- Optimizing this measure encourages seeking only some modes of the data distribution if the generator is not expressive enough
- The discriminator gets too confident on the generated samples passing no usable gradient to the generator (loss saturation)

Tweaking the generator loss function

- A solution that works in practice is to modify the part of the loss function of the discriminator used by the generator, so it does not saturate
- The reverse of the original loss function is used for the generator, putting all the gradient of the loss on the fake samples side
- Now the game is not a zero-sum game given that the optimization of generator and discriminator are different

$$\begin{aligned} \mathcal{L}_D &= \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \\ \mathcal{L}_G &= \mathbb{E}_{z \sim p(z)}[\log(D(G(z)))] \end{aligned}$$

Evolution of GANs

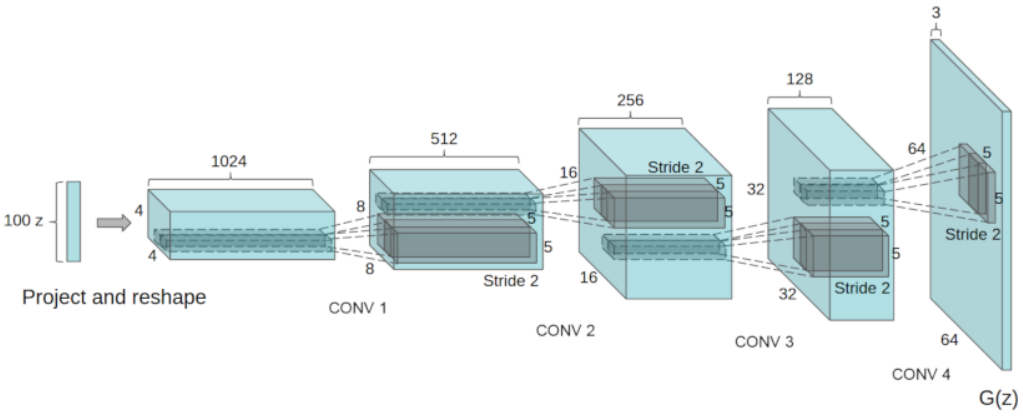
GANs Evolution

- Since the definition of GANs a lot of progress has been made
- Most of the original problems have been addressed by specific architectural choices for the generator and specific training decisions
- This has generated a lot of insight about what works and does not work for training GANs

DCGAN

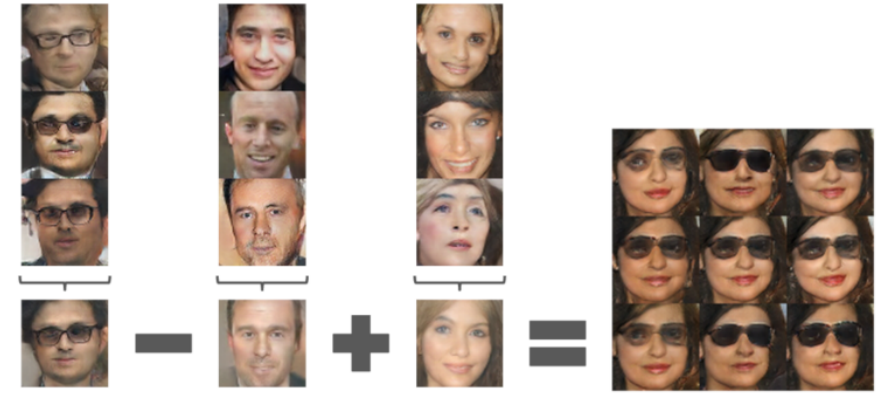
- This is the first GAN that consistently generated acceptable quality images
- Allowed to realize that many of the tricks of the trade of supervised deep learning with CNNs do not work for these models
- The architecture of generator and discriminator are defined symmetric
- Problems: Still unstable training, many specific hyperparameters, brittle architecture (high probability of collapse/divergence)
- Successive improvements have make training easier

DCGAN generator



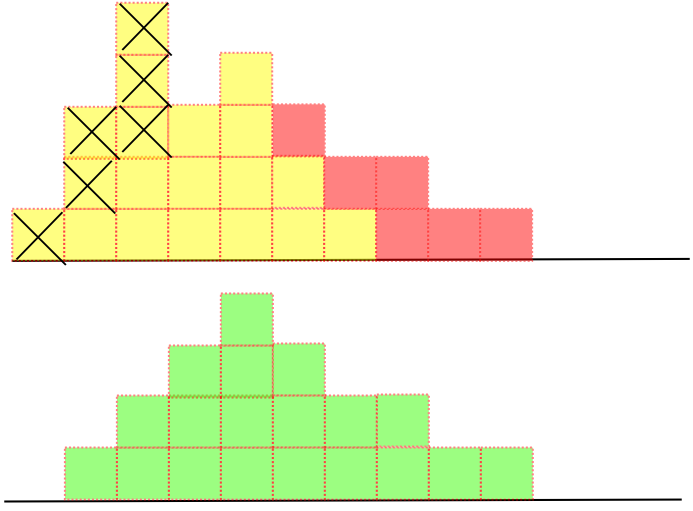
DCGAN smooth interpolation

DCGAN vector arithmetic



- There are alternative ways of measuring the similarity among the probability distribution of the real and generated data
- The Wasserstein distance is inspired on the earthmover distance from Optimal Transport
- Intuitively, in the domain of probability distributions this distance measures the cost of moving the probability mass of one distribution to obtain the other

Wasserstein distance



- This problem can be formulated as looking for the optimal distance function that maximizes the agreement between distributions
- We could search over a set parameterized functions f_W that is a lower bound of these functions (defined by a neural network) so:

$$\max_w \mathbb{E}_{x\sim P_a}[f_w(x)] - \mathbb{E}_{x\sim P_b}[f_w(x)] \leq \sup_{||f_L||\leq K} \mathbb{E}_{x\sim P_a}[f(x)] - \mathbb{E}_{x\sim P_b}[f(x)] = K \times W(P_a, P_b)$$

- This problem corresponds to a search in the space of k-Lipschitz functions (smooth functions)
- This is a problem that can be optimized using backpropagation

- The optimization problem is changed to:

$$\min_G \max_D \mathbb{E}_{x \sim P_D}[D(x)] - \mathbb{E}_{\tilde{x} \sim P_G}[D(\tilde{x})]$$

- Wasserstein distance correlates with samples quality
- WGAN is not too dependent on architectural or training choices, more robust than other formulations

- An improvement for the initial model of WGAN consists in to add a penalty term to the objective function (regularization)
- A 1-Lipschitz function must have everywhere gradients with norm at most 1
- We can try to force that by adding a penalty to the GAN objective function

$$\mathbb{E}_{x\sim P_D}[D(x)] - \mathbb{E}_{\tilde{x}\sim P_G}[D(\tilde{x})] + \lambda \mathbb{E}_{\tilde{x}\sim P_{\tilde{x}}}[(||\nabla_{\tilde{x}} D(\tilde{x})||_2 - 1)^2]$$

- This constraint is maintained sampling points $\tilde{x}$ in straight lines between samples from P_D and P_G

- SN-GAN introduces spectral normalization in the layers to maintain the Lipschitz property of the functions computed by the discriminator
- This introduces a more theoretically founded justification to the architecture
- The spectral norm of a matrix is defined as:

$$\sigma(A) = \max_{||x||_2 \neq 0} \frac{||Ax||_2}{||x||_2}$$

- The solution corresponds to the first singular value of A

- If the Lipschitz norm of the activation function is equal to 1 we can use the inequality:

$$||g_1 \circ g_2||_{Lip} \leq ||g_1||_{Lip} \cdot ||g_2||_{Lip}$$

- We can approximate the norm of the network as:

$$\begin{aligned} ||f||_{Lip} &\leq ||W^{L+1}h_L||_{Lip} \cdot ||a_L||_{Lip} \cdot ||W^L h_{L-1}||_{Lip} \cdots ||a_1||_{Lip} \cdot ||W^1 h_0||_{Lip} \\ &= \prod_{l=1}^{L+1} ||W^l h_{l-1}||_{Lip} = \prod_{l=1}^{L+1} \sigma(W_l) \end{aligned}$$

- This means that if for all the layers its parameter matrix W is normalized by its spectral norm $\sigma(W)$ the spectral norm of the function computed by the network is bounded by 1

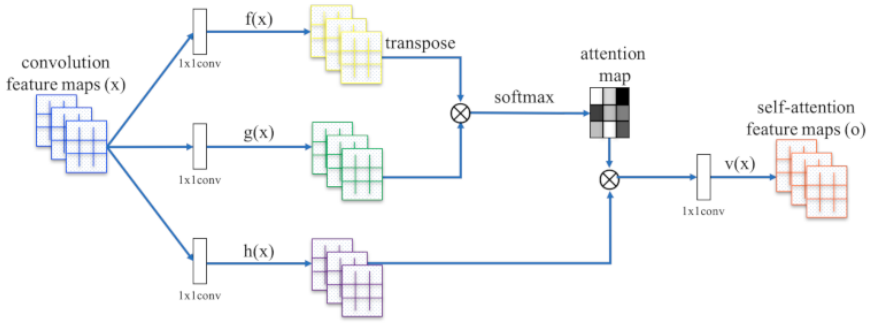
- To compute the eigenvalues of a matrix is expensive if we want to obtain the exact value, it involves computing SVD that is $O(n^3)$
- The first eigenvalue can be approximated using the power iteration method
- This method starts with a random vector that is multiplied by the matrix and then normalized, this is repeated for a number of steps

$$v_{k+1} = \frac{Wv_k}{||Wv_k||} \; ; \lambda = \frac{Wv_k \cdot v_k}{v_k \cdot v_k}$$

- In practice, for this task, just one iteration of this procedure obtains an adequate approximation

SAGAN: Self Attention

- Introduces self attention between convolutional layers for the generator and the discriminator
  - Features computed by the convolutional filters are passed through 1x1 convolutions and are used as query, key and values through the attention later



BigGAN (Google)

- BigGAN is a class conditional network that builds on all the GAN improvements (Self Attention + Spectral Normalisation)
- Use of residual networks, improved regularization
- . . . more computational power (more GPUs):
  - Increased batch size
  - Deeper model
  - More convolutional filters
- Truncation Trick:
  - During training the samples for the generator are from a normal distribution with variance σ^2
  - During sampling the values over a threshold $[-c, c]$ are truncated (resampled to a value inside the limit)
  - This improves quality at the expense of variety

BigGAN: Architecture

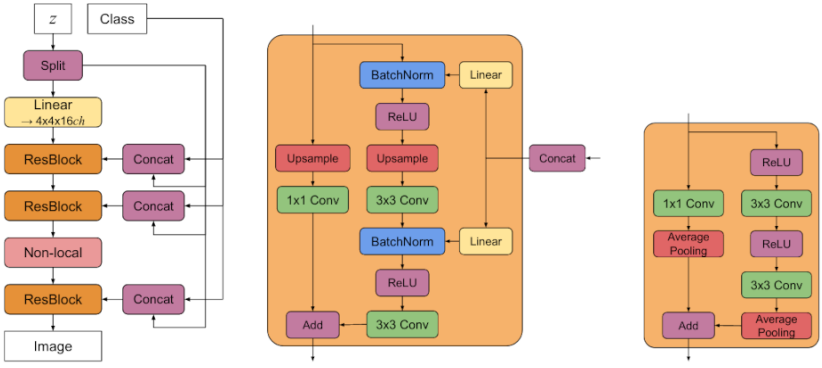


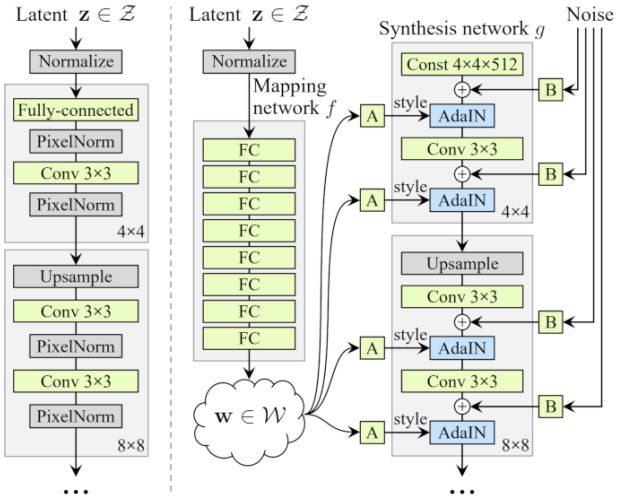
Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

BigGAN: Results (512×512)

StyleGAN family (NVIDIA)

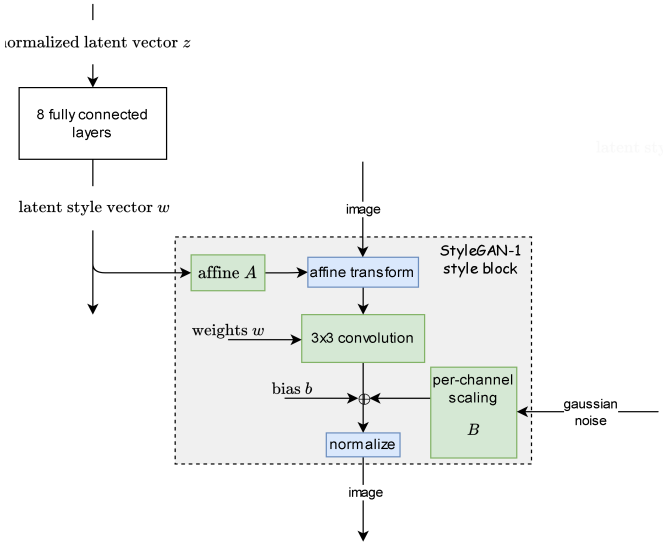
- StyleGAN departs from the usual architecture for the generator introducing a control of how the latent code modifies the image
- The generator starts with a constant learned code
- The random samples are passed through a fully connected network and feed to the generator at different resolutions (style vectors)
- Noise is added at different resolutions to improve quality (reduces blurred details)
- Adaptive Instance Normalization: an affine transformation that normalizes the examples according to the style vector
- Very good for homogeneous domains (faces, animals, landscapes, buildings...)

StyleGAN: Architecture



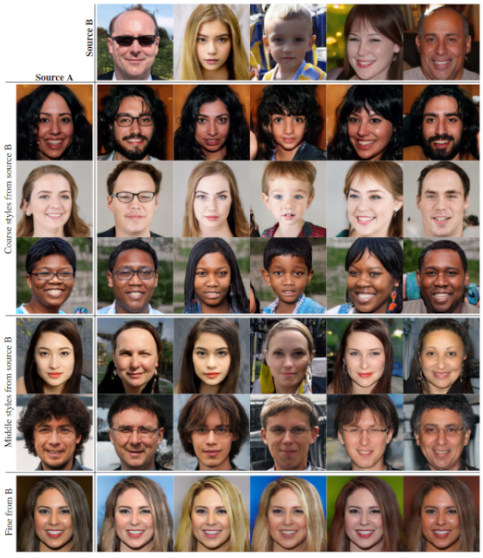
(a) Traditional (b) Style-based generator

Adaptive Instance Normalisation (AdaIN)



StyleGAN: Results

StyleGAN: Mixing different latent vectors at different levels

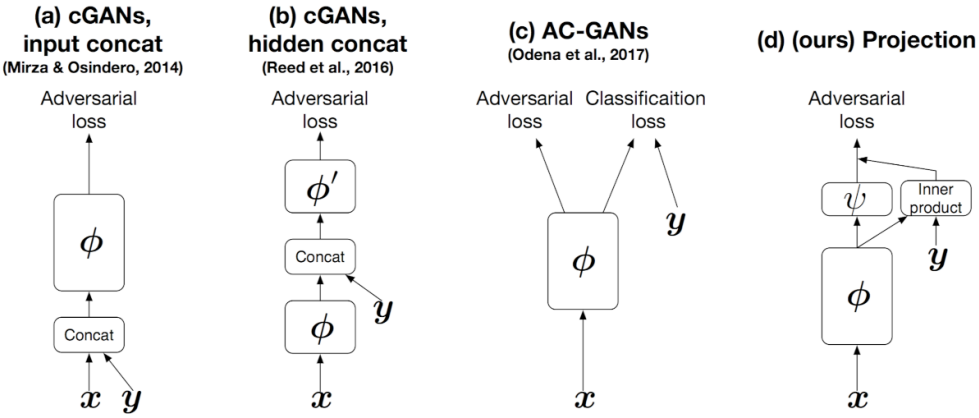


Conditional GANs

Conditional GANs

- We can use labels to condition GANs for generating samples from specific classes (this also can be done with previous models)
- Conditioning is not limited to labels
- Using different information as conditioning we can solve different tasks:
  - Image to image translation obtaining an image that *translates* pixels (colorization, semantic map generation, style transfer, super resolution...)
  - Text to image generation
  - Image inpainting/outpainting

Class conditioning



This Python Notebook shows examples of different GANs

- GANs Notebook (click here to open the notebook in colab)

If you download the notebook you will be able to use it locally (run jupyter notebook to open the notebooks)

BSC AI Factory

AI Factories

Co-funded by the European Union

Plan de Recuperación, Transformación y Resiliencia

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101234399 and Spain, Portugal, Romania and Türkiye. The JU receives support from the European Union's Horizon Europe Programme.

BSC AI Factory Partners

Affiliated entities