6. Generative Adversarial Networks (GANs)

INTRODUCTION

Generative adversarial networks (GANs) are deep neural architectures composed by two networks, set in opposition one against the other (thus the "adversarial"). Ian Goodfellow and other scientists in his group at the University of Montreal (CA) first proposed in 2014 the GAN architecture to investigate the ability for an artificial NN to create realistic data or even brand-new representations to reality like a human artist would do. GANs belong to the class of generative models that aspire to represent a distribution of a dataset from a statistical point view and develop methods to sample new instances from the distribution. Ultimately, generative models tries to reproduce a probability distribution through a statistical model driven by a probability density function p. The model provides an explanation of the variability of the dataset and can be used to predict/create a new data, not included in the original dataset but consistent to it. Traditionally, the estimation of the probability distribution of a dataset, provided that the parametric model is known, can be performed by data regression/fitting. In other words, the generative model learns the probability density function of the data (p_{data}) from the dataset approximating it by probability density function of the model p_{model}. Again, a new sample might be generated using p_{model}, this featuring a statistics adherent to original probability density p_{data}, which still remains however unknown.

However, things complicate when the model is unknown and data must be used to estimate in bundle the statistical model (e.g. multi-dimensional non-linear function) and the corresponding probability density (i.e. function parameters). This is just the field of search where ANN demonstrated nice abilities. GAN have proposed as a methodology to learn the characteristics of a data distribution where the generative model is unknown. In GAN domain, the generation model, called generator, implemented through a multilayer deep NN, is coupled to another multilayer deep NN called discriminator. The generator G performs a mapping to data space from a prior input variable. Namely, this consists of randomly sampling a low-dimensional white-noise distribution (input variable) usually called the 'root' of the generator. This variable undergoes progressive transform into the generator up the high-dimensional target data. The discriminator D is essentially a neural binary classifier that output a single scalar (Fig. 1).

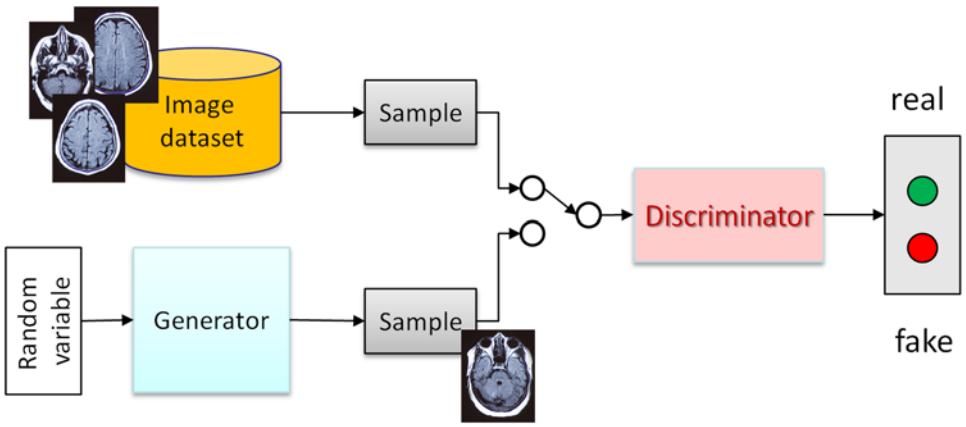


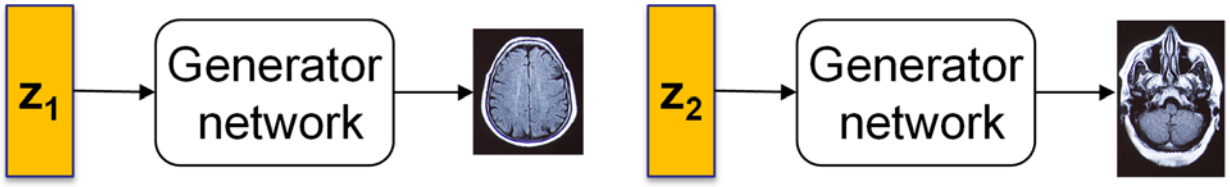
Figure 1. General topological schematics of a GAN.

The learning mechanism is based on a competitive (non-cooperative) two-player game between generator and discriminator, so that if one wins the other loses. In the game, the discriminator attempts to separate real data from virtually generated ones, assumed fake. Meanwhile the generator attempts to trick the discriminator by improving better and better the quality and the likelihood of the produced data during learning. The GAN converges when the discriminator and the generator reach the so-called Nash equilibrium. A Nash equilibrium will occur when one player will not change its action regardless of what the opponent may do. By construction, the training estimates in bundle two parameters sets (neural weights), one of the generator network and one of the discriminator networks. Again, as in the domain of neural network training, the topology of both networks must be provided a priori. The GAN training exploits a specific cost (loss)

function to drive and constraint the weight estimation from the data. The setup of the GAN requires also the definition of the random source and the dimensionality of the root for the generator. The quality, the variety and the number of training data affect as usual the performance of the training. We will see next that the GAN training is based on minimax optimization of the loss function.

Generator network

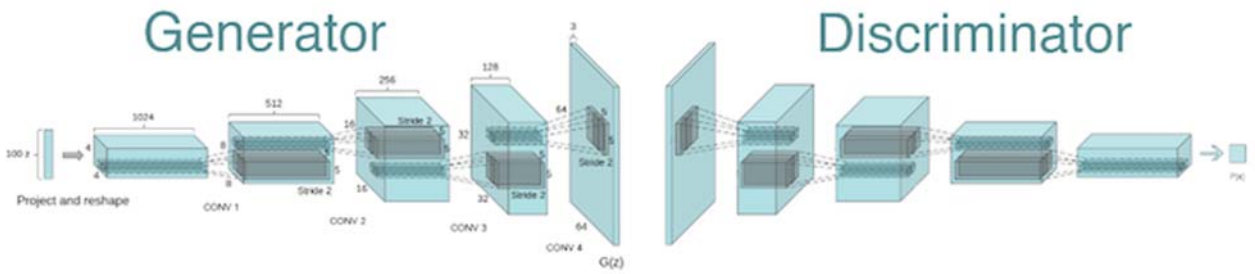
The generator network is devoted to data reconstruction from the root, namely a random variable. Now we look into how a generator creates images before learning how to train it. First, we sample some noise *z* using a normal or uniform distribution. With *z* as an input, we use the generator *G* to create a new data. Let us here consider the bioimage space for the data. By selecting a realization z_1 (e.g. 0.2, 0.04, -0.7, …) extracted from $z \sim N(0,1)$, the generator map z_1 in $I_1=G(z_1)$. A different realization z_2 (e.g. -0.6, 0.9, -0.3, …) will be mapped into a different image $I_2=G(z_2)$. In GAN paradigm, *z* represents the latent features of the images generated, for instance the gray-scale, the shape, the tissue density and many other characteristics. These are a priori unknown and we expect the training process to learn them. From an operational point of view, the generator can be regarded as a mechanism to reconstruct an image starting from a low-resolution version of such image. In the limit, the low-resolution image contains only the main descriptive features. In mathematical notation, this goes as up-sampling.



As we described in the previous note, decoding from input z domain to output domain can be regarded as an up-sampling operation. Considering image ouput, up-sampling means computing each output y_{ij} (image pixels) from inputs by an interpolation map (nearest neighbor, linear, bi-linear,..). This map replicates a one-to-many relationship, say one pixel in the input image will be responsible of many pixels in the output image. The result of the interpolation depends on the map and on the reciprocal positions of the input and output image regions, featuring locality. Practically, this can be done by using image deconvolution, or 'inverse' convolution. Deep learning approach to optimal up-sampling does not use a predefined interpolation method and implements the deconvolution by means of transposed convolutional filtering.

Discriminator network

For the discriminator, a feed-forward multilayer network, that must opportunely synthesize the input data into a final two-class output, can be taken into account. The straightforward approach is represented by using a convolutional network with terminal flatten layer, dense layer and single logistic neuron in the output. Therefore, the architectures of the generator and discriminator can be summarized as we did in the following picture.



GAN training

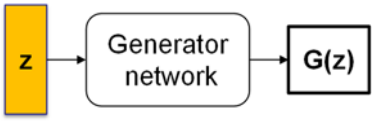
In the GAN domain, the concept of two-player game grounds on the competition between the generator and the discriminator. In this match, the generator network tries to fool the discriminator by producing real-

looking images while the discriminator tries to increase its ability to separate between real and fake images. According to that, the training must step-by-step improve the generator in such a way to successfully trick the discriminator so that at the end it is generating images that look like image originating from the training set. Therefore, as the GAN includes the two neural networks, the training estimates two parameter sets namely:

θ_g: parameters of the generator network

θ_d: parameters of the discriminator network

jointly. At each iteration, the training computes the discriminator output *D* (bounded between 0 and 1) as a function of input that can be *x* (real data) or alternatively *G*(*z*), namely the data simulated by the generator and evaluates the corresponding two log probabilities as:



$$E_{x\sim p_{data}} log D_{\theta_d}(x)$$

$$E_{z\sim p_z} log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$

where p_z is exactly p_{model}. The first one is predicting that real data is genuine (assuming the x belongs to the model p_{data}) whereas the second one is predicting that the data generated by *G* using the current *z* (assuming the z belongs to the model p_z) is not genuine. Ian Goodfellow (2015) proposed to exploit such two quantities into the minimax game formulation, which takes its root from the principle of Nash equilibrium into a competitive environment. In simpler terms, when two players (*D* and *G*) are competing against each other (zero-sum game), and both play optimally (minimax strategy), assuming that their opponent is fair, the outcome is predetermined and none of the players can change it (Nash equilibrium).

The minimax problem reads thus as:

$$min_{\theta_g} max_{\theta_d}\left[E_{x\sim p_{data}} log D_{\theta_d}(x) + E_{z\sim p_z} log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

where the discriminator wants to maximize the objective such that $D_{\theta_d}(x)$ is close to 1 (real) and $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ is close to 0 (simulated/fake), while the generator wants to minimize the objective such that $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ is close to 1 (discriminator is tricked into thinking generated $G_{\theta_g}(z)$ is real). It can be demonstrated that the optimal solution requires that p_z replicates p_{data}.

This mechanism of competitive optimization can be further detailed by a chart (Fig. 6). Given that the input to the discriminator is x (real data) then $D_{\theta_d}(x)$ tries to be close to 1. Whether the input is $G_{\theta_g}(z)$, the discriminator tries to make $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ close to zero. Concurrently, the generator tries to fool the discriminator to put $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ close to 1.
According to the minimax optimization, gradient-based error backpropagation is implemented, alternating one mechanism that drives the estimation of the weights of the discriminator and the other one that drives the estimation of the weights of the generator as depicted in the next picture (Fig. 6). It can be shown that the expectation value can be approximated by a discrete sum over a finite batch of *m* samples at each iteration.

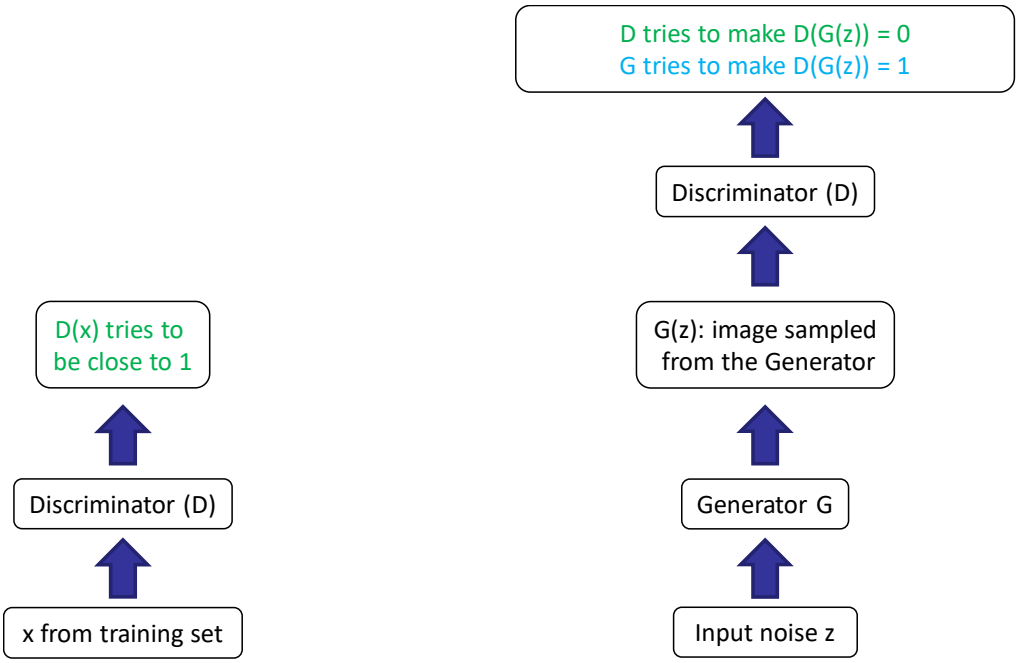


Figure 6. Competition between generator and discriminator.

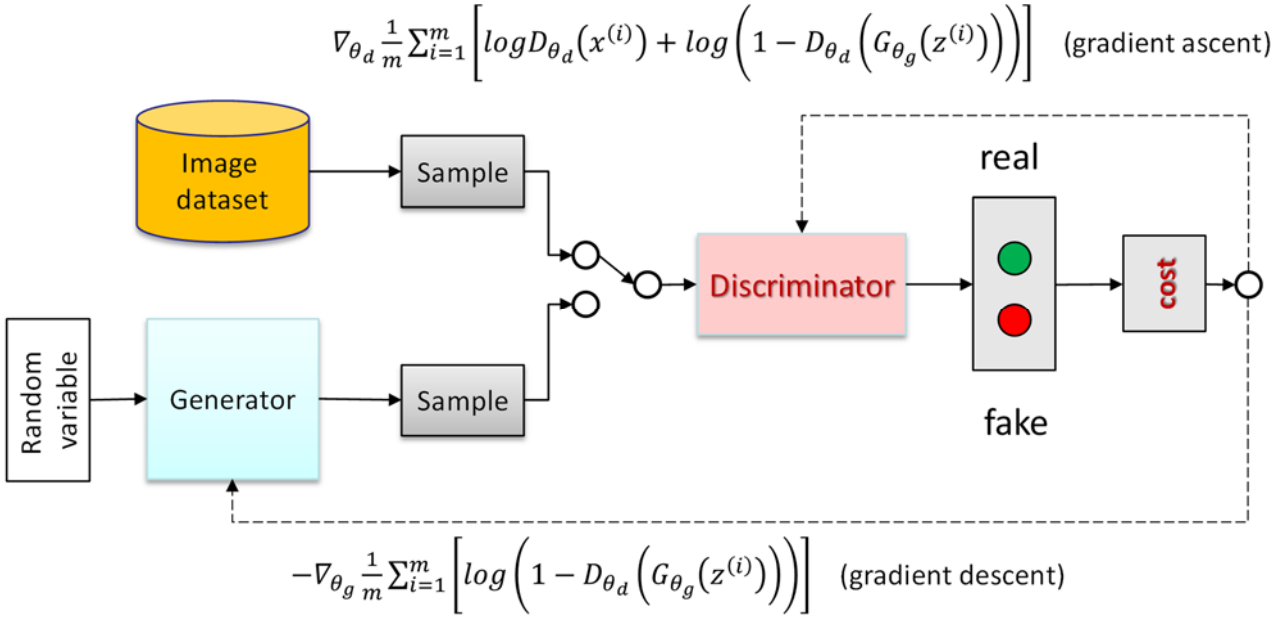
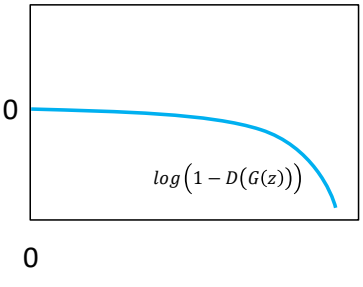


Figure 7. Minimax training mechanism of the GAN assuming a batch of *m* samples using the discrete approximation of the expectation. The negative sign is consistent with the fact that we are moving along the loss function in the counter direction with respect to the gradient vector.

Some implementation comments about the optimization of the generator are to be reported though. We know that the optimization implies a gradient descent on the generator by:

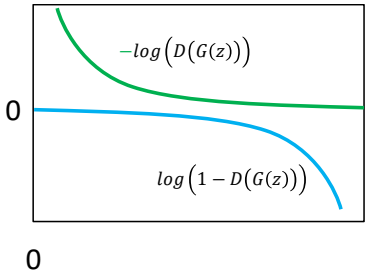
$$min_{\theta_g} E_{z \sim p_z} log\left(1 - D_d\left(G_{\theta_g}(z)\right)\right)$$



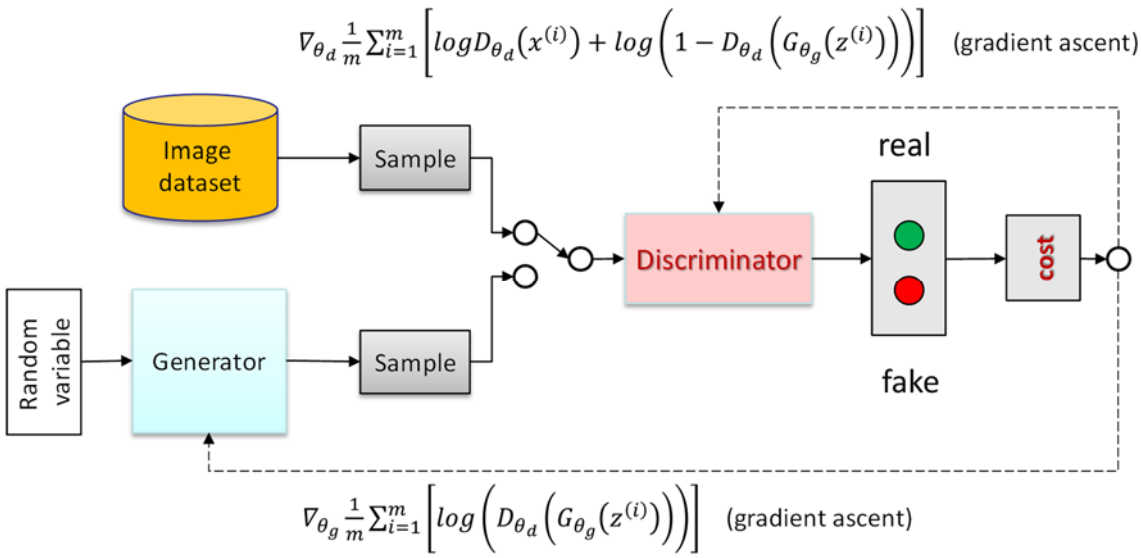
Basically, the gradient signal is dominated by the region where sample is already good ($D_d\left(G_{\theta_g}(z)\right)$ ~1). But the aim of the generator is to learn when

the sample is likely fake ($D_d\left(G_{\theta_g}(z)\right)$ ~0). However, in such a region, the gradient is small and this results into a weak update being therefore the minimization over the parameters of the generator dominated by the maximization over the parameters of the discriminator. One practical solution consists in reversing the objective by means of a gradient ascent on a different generator objective as:

$$max_{\theta_g} E_{z\sim p_z} log\left(D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$



Therefore, instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong. The formal objective is exactly the same as the earlier but we have now higher gradient signal for bad samples. The final training structure can be summarized thus in the next picture



As the generator improves with training, the discriminator performance gets worse because the discriminator cannot easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. This means that the discriminator feedback gets less meaningful over time. This can pose convergence issues. For example, if the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse. This is just to remark that jointly training two networks is challenging and can be unstable. Choosing objectives with better loss landscapes might help. This training approach in an active area of research now in the field of GANs.

Regarding the training of a traditional GAN, it is correct to say that the labels of the real dataset are defined a priori to true (1) but it is also correct to say that by default the element label (synthetic data) output from the generator is false (0). During the training, the discriminator is warned about which of the two it is receiving, therefore in the loss function the label is known. For this reason, discriminator training can be considered supervised (see class presentation where if the element comes from dataset D (x) -> 1 while if it comes from generator D (G (z)) -> 0). The discriminator weights are updated by maximizing the gradient of f1 = log (D (x) + log (1- D (G (z))) clearly calculated with respect to the discriminator weights i.e. df1 / dw_d while the generator weights are fixed. For the generator, f2 = log (1- D (G (z))) with respect to w_g is maximized and the discriminator weights are not touched. With this maximization the generator tries to force the discriminator to produce D (G (z)) = 1 in conflict with the previous criterion for estimating the parameters of the discriminator. The generator does not know what the reference is for the element that is generating at a particular moment while the discriminator does, but overall supervision is also valid for the generator (

indirectly). There is a proof that, in an ideal training, as one approaches the convergence the generator improves more and more and it happens that the discriminator confuses its input more and more. The discriminator knows the label of the input he is receiving at that moment but the output produced is wrong. When a real element enters from a certain point onwards in the training it starts to make mistakes so D (x) will not always be 1 but sometimes it will be 0. At convergence the probability of D (x) = 1 will be 0.5 therefore identical to the probability of D(x) = 0. The same thing happens for D (G (z)) but in the opposite sense. At the beginning D(G(z)) = 1 has a very low probability and during training the probability begins to rise and sometimes D(G(z)) will be = 1. As the probability D (G (z)) = 1 goes up we get closer to convergence. For balancing with D(x), the probability D(G(z)) = 1 will come to 0.5. At the end, therefore, the discriminator has a 50% chance of making a mistake on the real element and 50% of making a mistake on the synthetic element produced by the generator. In this case, the generator is said to have learned the statistics of the training set data. In the real case, it is actually necessary to monitor the two probabilities so as not to allow them to differ too much from 0.5. From previous experiences, if you let the training go on, the update of the generator weights loses its effectiveness because the discriminator is giving it more and more completely random feedback. After a while, it happens that the generator loses grip with the statistics it had learned and can collapse to a subset of the statistics or even go completely wrong.
In conclusion, the GAN approach does estimate the model of the data distribution using a competition between two networks, namely the generator and the discriminator. Once the weights are computed, the discriminator can be detached from the generator. This last can be then used to generate new data inputting z values at random.

GAN variations

Research in the field of GAN is still in progress addressing both innovative architectures and training strategies. One example is the Progressive Growing GAN (proGAN) applied to image generation (Karra et al., 2017), which rests on the gradually increase of the resolution of both the generator and the discriminator. The training starts with a low resolution and gradually add new layers so that model increasingly refine the image details as training progresses. This technique allows the GAN to train more quickly than comparable non-progressive GANs and produces higher resolution images. Another relevant contribution named conditional GAN (cGAN – Mirza et al., 2014) consists of the modification of the generator in its learning paradigm extending the random noise vector z to a structured input coherent with true data x so that the input to the generator becomes (x, z), with the optimization problem reformulated as:

$$min_{\theta_g} max_{\theta_d} \left[E_{x \sim p_{data}} log D_{\theta_d}(x) + E_{(x,z) \sim p_{(x,z)}} log \left(1 - D_{\theta_d} \left(G_{\theta_g}(x, z) \right) \right) \right]$$

This modification leads to feed both the generator and discriminator with true data x so that the conditional GAN will explicitly learn a tag variable (random variable z) corresponding to a data domain

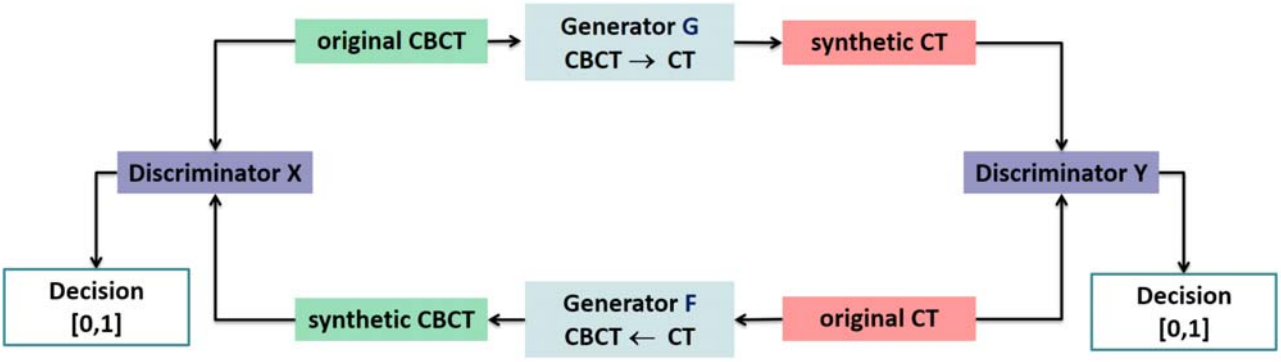
Cyclic coherency GANs have been proposed to remove the dependency on paired true and simulated data required in the training of previous GAN models. In the very last years, cyclic GANs have become the focus of research in the field of biomedical images, with particular focus on image-to-image translation. In order to describe cyclic GANs, we carry out a specific example in radiation therapy of mapping cone beam CT to CT images. As said above, cyclic GAN do not require paired training data, but only a set of unpaired (not coming from the different patients but also being not corresponding slices in the scan) images CBCT and CT images. The basic architectural idea of cyclic GANs is that there exist two generators: the first one generator should be able to map the original CBCT to a synthetic CT (CBCT similar to the original CT), the second one should be able to map the original CT to a synthetic CBCT (CT similar to the original CBCT). Each generator will be coupled to the corresponding adversarial discriminator. The overall adversarial loss will be therefore composed by two objective functions as:

$$L_Y(G, D_Y, X, Y) = \left[E_{y \sim p_{data}} log D_Y(y) + E_{(x) \sim p_{(x)}} log\big(1 - D_Y(G(x))\big) \right]$$

where x is the original CBCT image, G(x) is the corresponding synthetic CT produced by the generator G and y is the original CT image (let us recall that the synthetic CT and the original CT cannot be paired), and

$$L_X(F, D_X, X, Y) = \left[E_{x \sim p_{data}} log D_X(x) + E_{(y) \sim p_{(y)}} log\big(1 - D_X(F(y))\big)\right]$$

where now the generator F(y) maps the original CT image y back to the CBCT domain and the discriminator X compares the original CBCT x with the synthetic CBCT.



However, the adversarial losses alone have been acknowledged not enough to produce high quality images, according to the condition relaxing the correspondence between the two inputs to the discriminator (unpaired paradigm). Actually, it enforces only that the generated output is of the appropriate domain, but does not enforce that the input and output are the same image. A generator network, on the other hand, may map the same set of input images to any random permutation of images in the target domain, with any of the learned mappings producing an output distribution that matches the target distribution. For example, a generator that output an image y that was an excellent example of domain Y, but looked nothing like x, would do well by the standard of the adversarial loss. Thus, to reduce the space of possible mapping functions, the concept of cycle consistency is formulated. It is based on the assumption that if you convert an image from one domain to the other and back again by feeding it through both generators in sequence, you should get something similar to what you put in. The two cycles are called respectively forward cycle consistency and backward cycle consistency according to the following mappings:

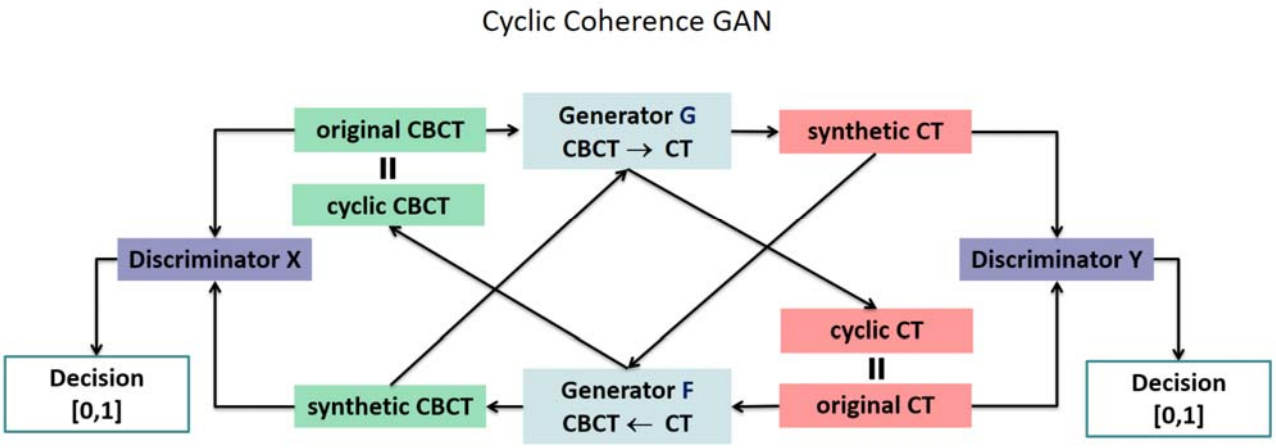
$$x \rightarrow G(x) \rightarrow F(G(x)) \approx x$$
$$y \rightarrow F(y) \rightarrow G(F(y)) \approx y$$

recalling that, in the target example,:

- x is the original CBCT
- G(x) is synthetic CT
- y is the original CT
- F(y) is the synthetic CBCT

The combination of the two cyclic losses, named cycle consistency loss, is shown below:

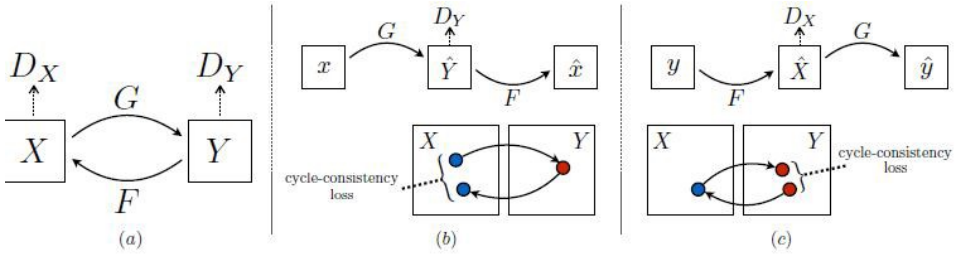
$$L_{cyc}(G, F) = E_{x \sim p_{data}}\big(F\big(G(x)\big) - x\big) + E_{y \sim p_{data}}(G(F(y)) - y)$$



Now, we formulate the complete objective function for the cyclic coherence GAN as:

$$L = L_Y(G, D_Y, X, Y) + L_X(F, D_X, X, Y) + \lambda * L_{cyc}(G, F)$$

that can be solved by the traditional minimax optimization procedure. The factor λ controls the relative importance of the adversarial and cycle-consistency parts of the objective and is to be considered a hyper-parameter of the network. The overall objective can be better described looking at the three sketches where (a) represents the traditional adversarial losses L_X and L_Y, whereas (b) and (c) represent the consistency in the arc x (CBCT) and the consistency in the arc y (CT).



To summarize, GAN does not need a priori density function of the generator and learns how to generate from training data through 2-player game. However, the training is more unstable with respect to traditional FFNN and CNN. Intensive research is on-going to both evolve new architectures (cGan, ProGan, …) and improve quality and speed of the training.