The Magic of GANs (Generative Adversarial Networks)

Alex Movila

Competition is good for creativity, but training is difficult, unstable and resource intensive.

Youtube - Magic of GANS

[IASI AI] GAN Cool Implementations

AI Can Manipulate Video to Make Everybody Dance Now (Youtube)

Vid2Vid—Conditional GANs for Video-to-Video Synthesis (Youtube) (github)

Nvidia has created the first video game demo using AI-generated graphics (Youtube)

Progressive Growing of GANs for Improved Quality, Stability, and Variation (Youtube)

Look at These Incredibly Realistic Faces Generated By A Neural Network , Youtube (Style-GAN - NVIDIA)

Steve Buscemi + Jennifer Lawrence MASHUP on Youtube (Deepfakes with ReCycle-GAN)

DeOldify - Colorizing and Restoring Old Images with GAN (Self-Attention GAN**)**

Christie's sells its first AI portrait for $432,500, beating estimates of $10,000

Recycle-GAN: Unsupervised Video Retargeting (Creepy AI transfers facial expressions in videos from one person to another) (Youtube) (Recycle-GAN)

Unpaired Image-to-Image Translation using Cycle-Consistent GANs (Cycle-GAN) (Youtube)
AI Generates Images of a Finished Meal Using Only a Written Recipe

DeepMind AI can generate convincing photos of burgers, dogs, and butterflies (BigGAN)

Enhanced Super-Resolution GAN Remasters Max Payne (*ESRGAN) (Topaz AI Gigapixel)*

New machine learning algorithm breaks text CAPTCHAs easier than ever (only 500 real data samples needed)

[IASI AI] GAN Cool Implementations

Inside the World of AI That Forges Beautiful Art and Terrifying Deepfakes

Photo-Realistic Single Image Super-Resolution (generate hi-rez photo from low-rez)

Generative Visual Manipulation on the Natural Image Manifold (reshape , re-color designer objects using small cue lines)

AI portraits (upload your photo to generate your painted portrait)

Given a satellite image, machine learning creates the view on the ground

Understanding AttnGAN: Text-to-Image convertor (StackGAN) (StackGAN and HDGAN)

CariGANs: Unpaired Photo-to-Caricature Translation (uses 2 Cycle-GANs one for geometry and one for style)

GAN Paint (A neural network can learn to organize the world it sees into concepts - just like we do)

This clever AI hid data from its creators to cheat at its appointed task (Cycle-GAN clever failure - uses stenography to hide data in the image)

Nvidia researchers develop AI system that generates synthetic scans of brain cancer (data augmentation with GAN)

Amazing new AI program transforms photos into gorgeous anime artwork (Image translation)

Prisma App (transfer artistic style from another photo to yours)

FaceApp - How would AI color my hair? (make my face younger or older, male to female etc)

[IASI AI] The Magic of GANs (Generative Adversarial Networks)

Generative Adversarial Networks is the most interesting idea in machine learning in last ten years, Yann Lecun (Facebook AI Director)

What I cannot create, I do not understand - Richard Feynman (American theoretical physicist)

Adversarial training

Training a model in a worst-case scenario, with inputs chosen by an adversary.

- Both players are neural networks dueling each other – if one wins the other loses
- Worst case input for one network is produced by another network

CVPR 2018 Tutorial on GANs
GANs will change the world

The Magic of GANs - Why Generative Models?

We've only seen discriminative models so far

- Given an image **X**, predict a label **Y**

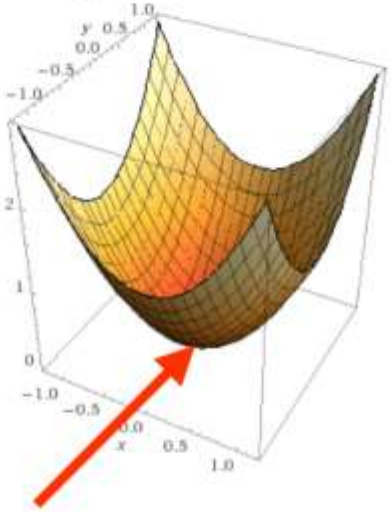
Generative models

- Can generate new images

Ian Goodfellow Conference Presentations
Ian Goodfellow: GAN (NIPS 2016 tutorial): Youtube, Paper

Adversarial Machine Learning

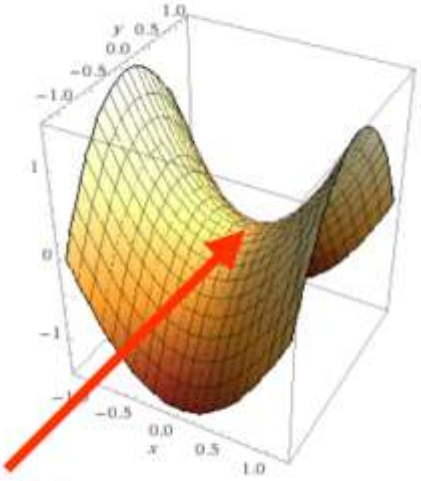
Traditional ML:
optimization



Minimum

One player,
one cost

Adversarial ML:
game theory

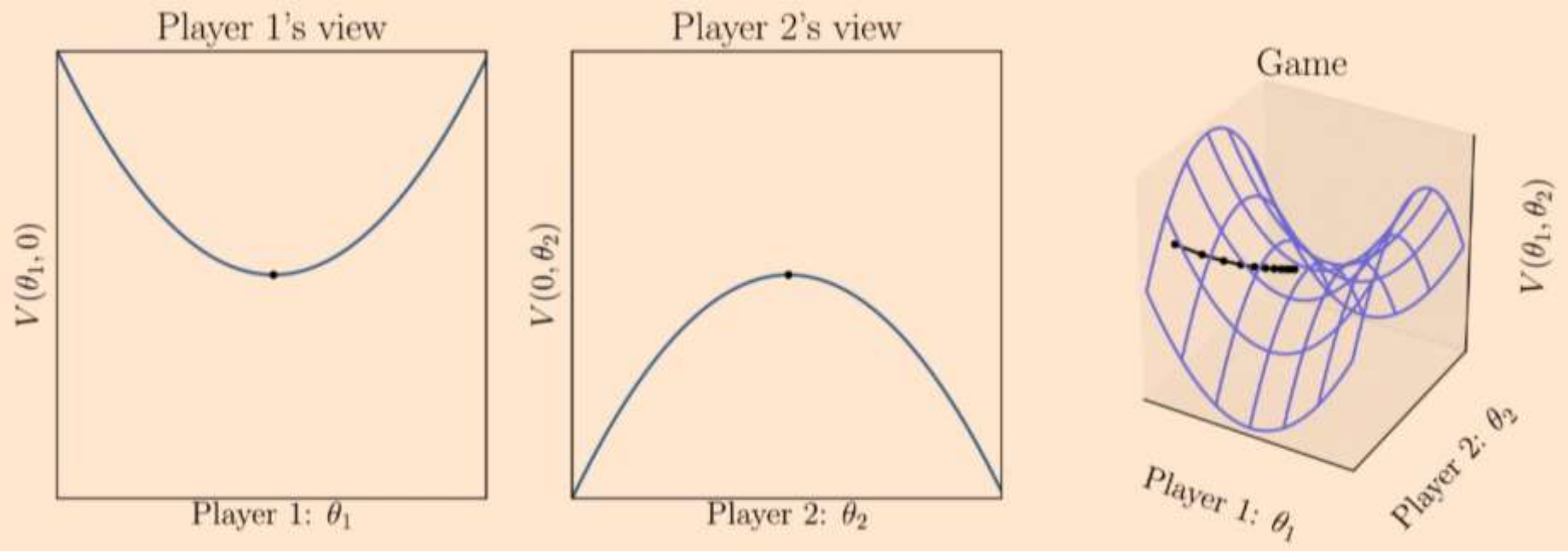


Equilibrium

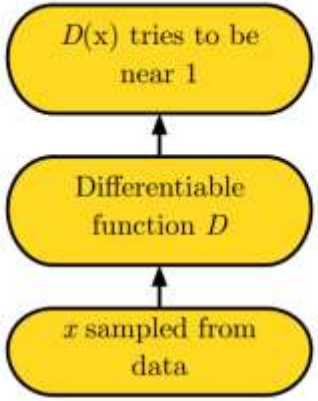
More than one player,
more than one cost

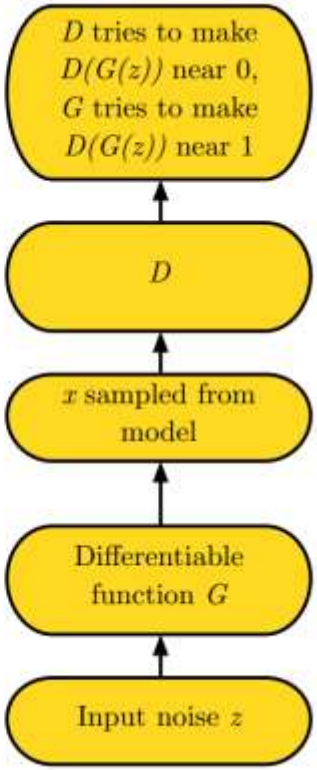
Game theory - Minimax game

Cost for player 2 is negative of cost for player 1 => one cost function only , player 1 wants to minimize it , player 2 wants to maximize it:



ADVERSARIAL NETS FRAMEWORK





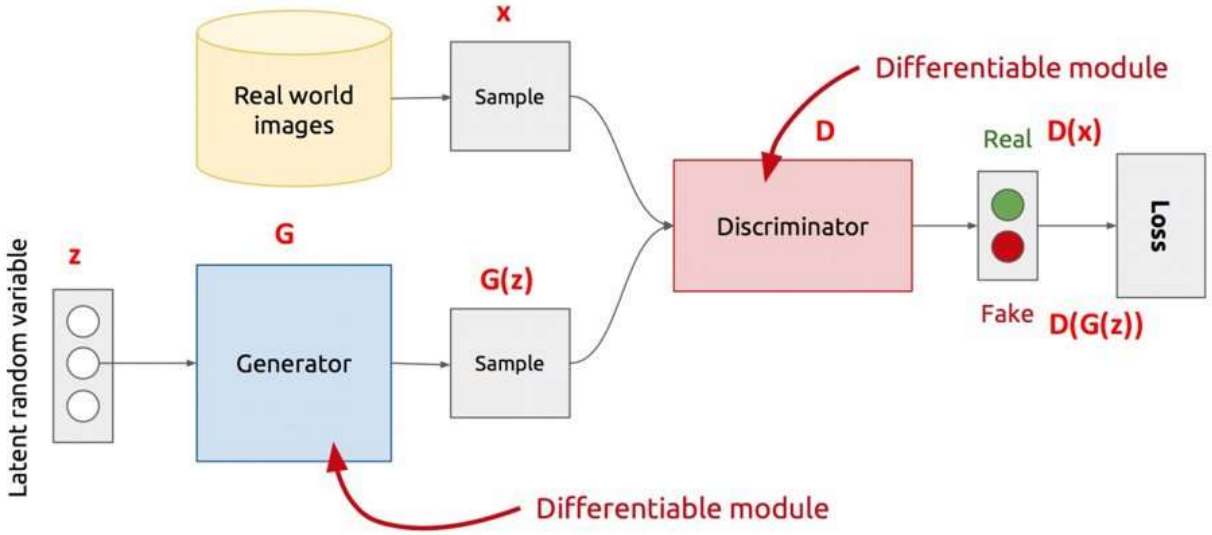
(Goodfellow et al., 2014)

Sometimes loss function cannot be expressed analytically. How do you write loss for a NN to compare how realistically is an image. Solution – use another NN to learn the loss function.

Note:
D & G have opposite objectives
G never sees directly the training data

Key thing to realize here is that GANs essentially are learning the loss function for you – which is really one big step closer to toward the ideal that we're shooting for in machine learning. And of course you generally get much better results when you get the machine to learn something you were previously hand coding.

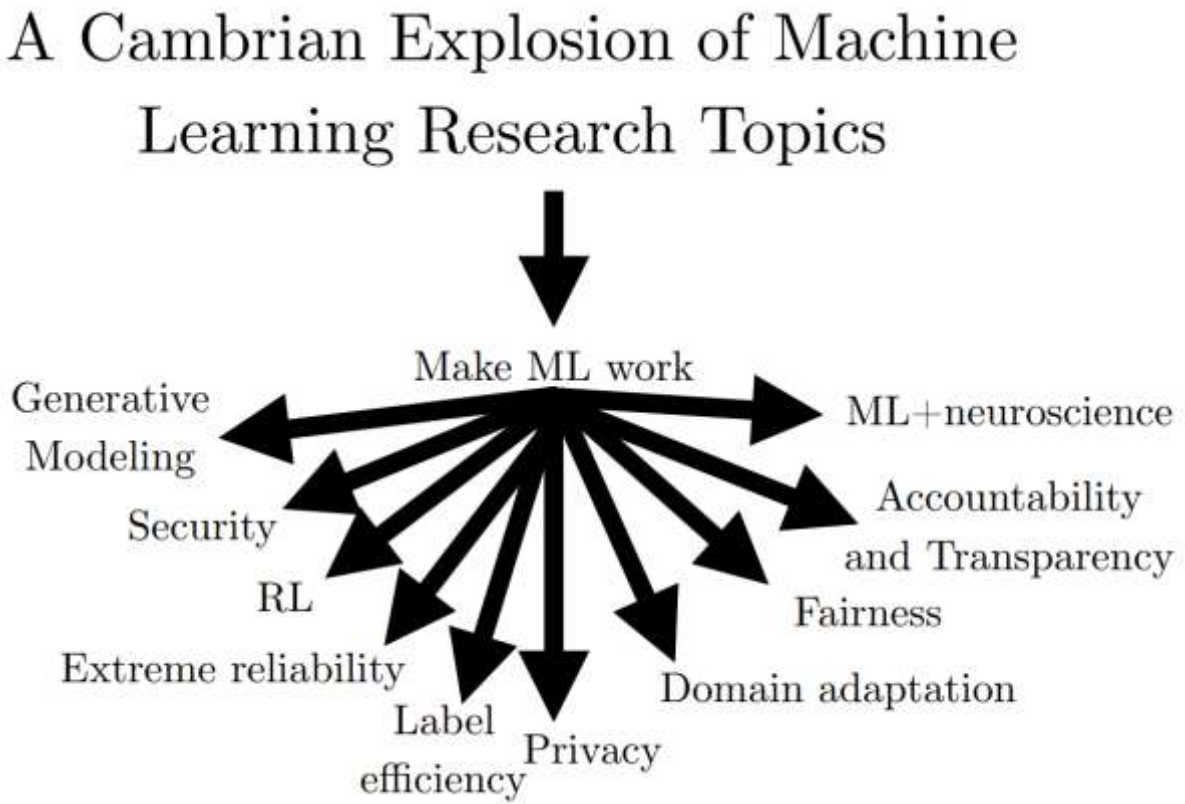
GAN ARCHITECTURE



Adversarial Training

- Generator: generate fake samples, tries to fool the Discriminator
- Discriminator: tries to distinguish between real and fake samples
- Train them against each other
- Repeat this and we get better Generator and Discriminator

- Z is some random noise (Gaussian/Uniform).
- Z can be thought as the latent representation of the image.



Adversarial Machine Learning - Ian Goodfellow , (Conference on Vimeo)

[IASI AI] ALTERNATIVE USE CASES

1) Realistic generation tasks

Auto texturize and render a game, generate items for fashion and design, aging faces, improve photos, add a smile etc.

2) Data augmentation

NVIDIA showed amazing example : they used GANs to augment dataset of medical brain CT images with different diseases and showed that the classification performance greatly improved

7) Missing data - Semi supervised learning

- can learn even when there is some unlabeled data,
- semi-supervised: use labelled data + many more unlabeled data

3) Domain adaptation

Convert a dataset to a different one for a new but similar domain; train in a simulation, test in the real-world

4) Data manipulation (use GANs with disentangled representation) (photos – add smile etc)

5) Adversarial training

Make a model more stable to adversarial attacks and improves its performance using smart data augmentation and regularization

6) Simulate possible futures for planning or simulated RL

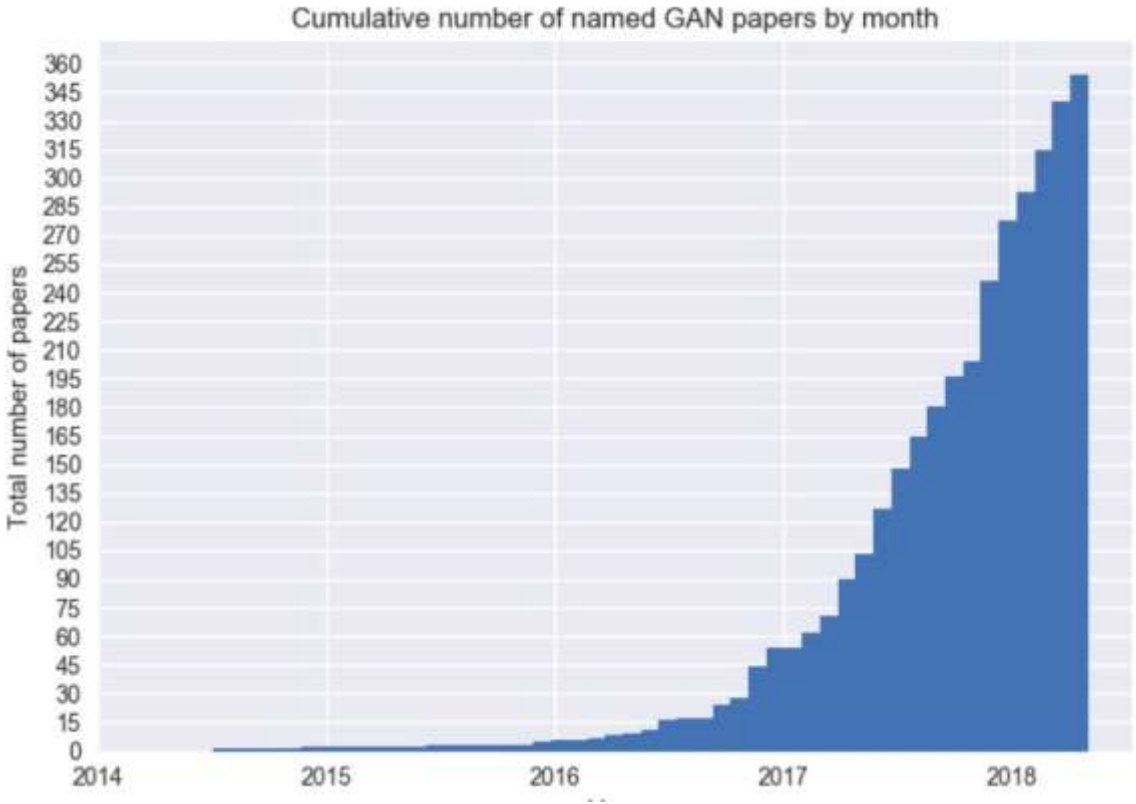
- autonomous cars – give me possible trajectories of other cars

8) Multi-modal outputs

- one to many input to output mapping (NN give alternative solutions)

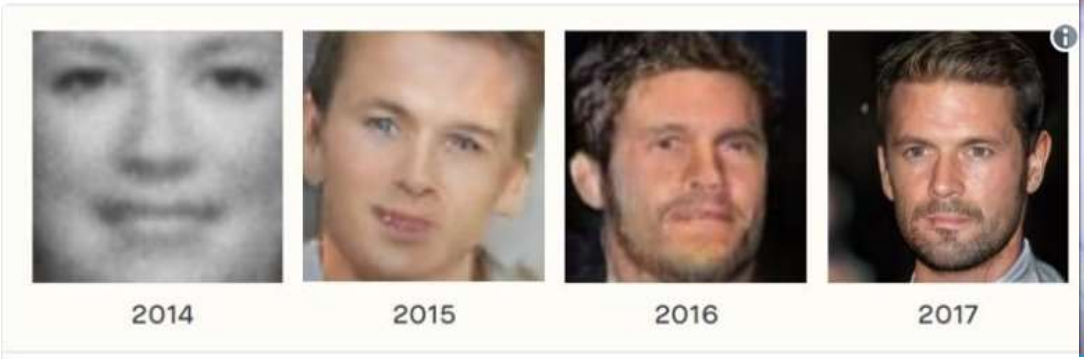
GANs beyond generation: 7 alternative use cases

TRACK GAN PAPER UPDATES AT THE GAN ZOO



The GAN Zoo (so many GAN types)
GAN-Timeline (evolution of GANs)
A list of papers on GANSs
Keras-GAN Implementations

4 YEARS OF GAN PROGRESS ON FACES (2014 – 2018)



Ian Goodfellow
@goodfellow_ian

4 years of GAN progress (source: eff.org/files/2018/02/…)

♡ 2,949 5:26 AM - Mar 3, 2018

ProGAN: How NVIDIA Generated Images of Unprecedented Quality (2018)

Malicious Use of Artificial Intelligence report

[IASI AI] 2 YEARS OF PROGRESS ON IMAGENET

Odena et al
2016

Miyato et al
2017

Zhang et al
2018

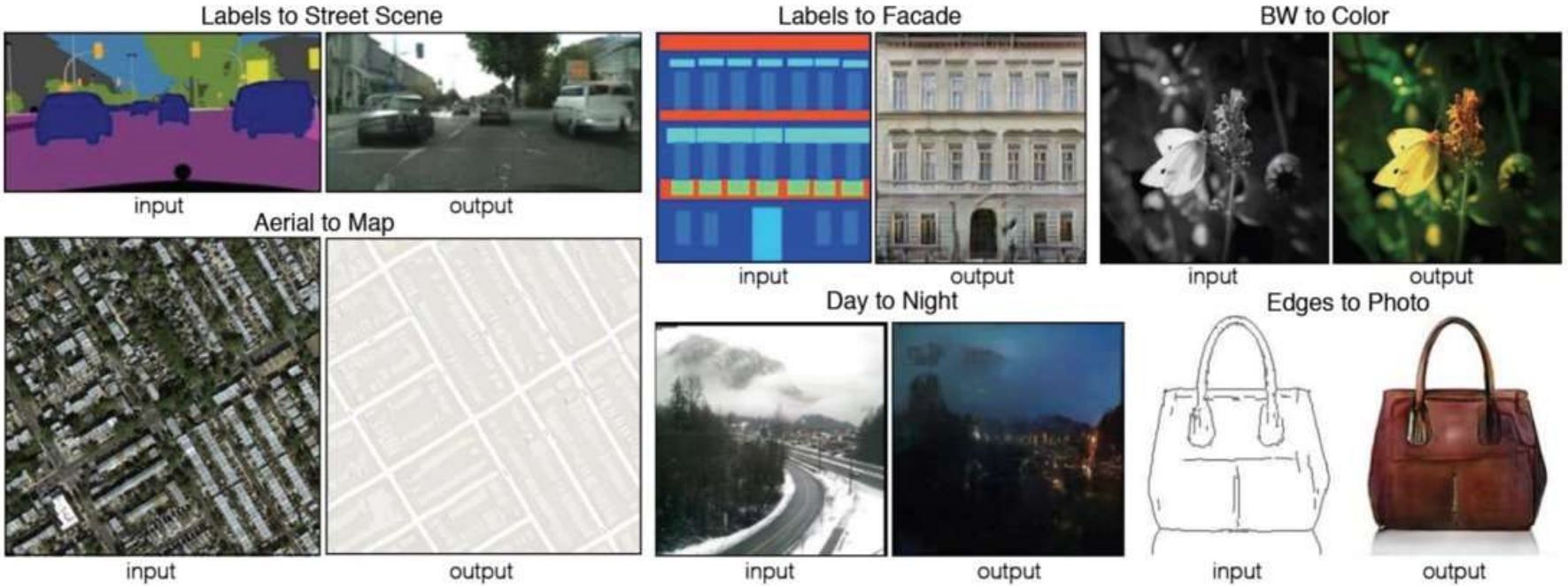
BIGGAN - STATE-OF-THE-ART ON IMAGENET (2018)

" GANs benefit dramatically from scaling, and train models with two to four times as many parameters and eight times the batch size compared to prior art"

Samples from BigGAN, from Google

[IASI AI]

CGAN APPLICATIONS - IMAGE TO IMAGE TRANSLATION WITH PIX2PIX (2016)

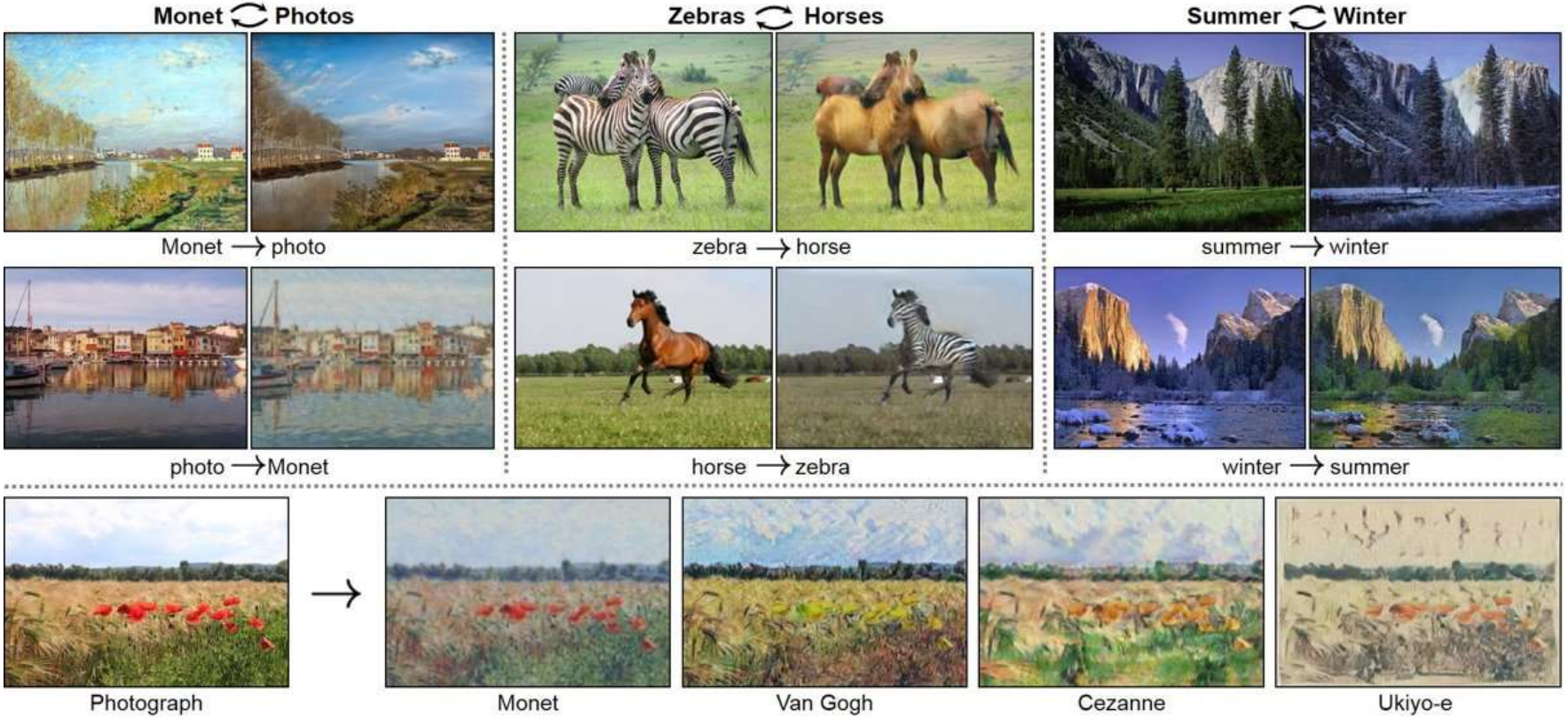


Pix2Pix demo, Real-time Pix2Pix , Colab Code

Pix2Pix Twitter-driven research

Image-to-image translation with conditional adversarial networks (Paper)

CYCLEGAN – STYLE CONVERSION



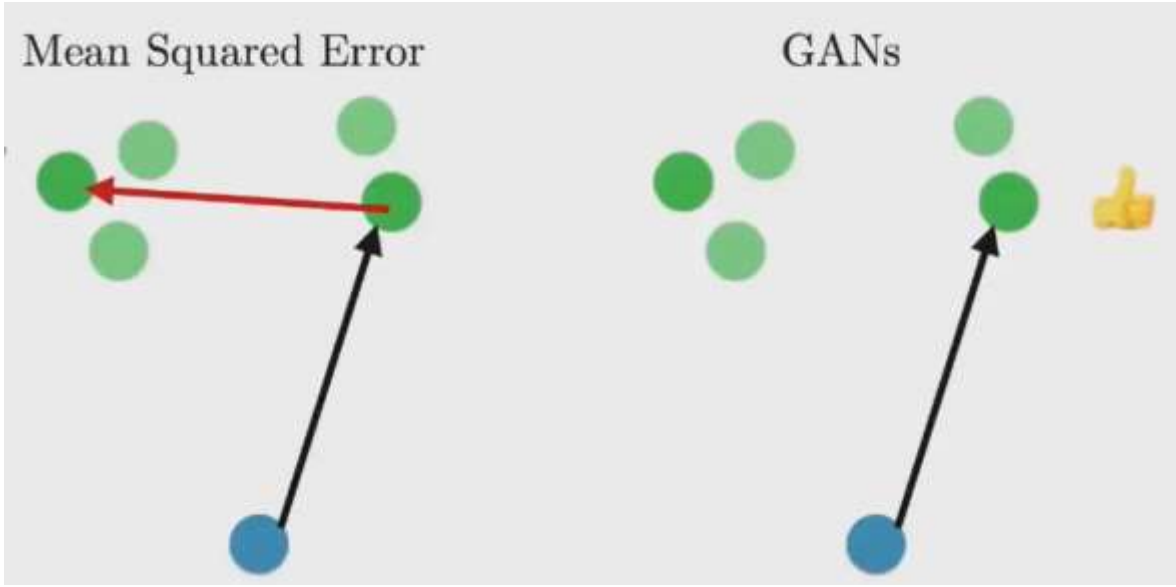
Self Supervised learning: generative approaches

[IASI AI]

Creative Adversarial Networks (CANs)

What are Creative Adversarial Networks (CANs)?

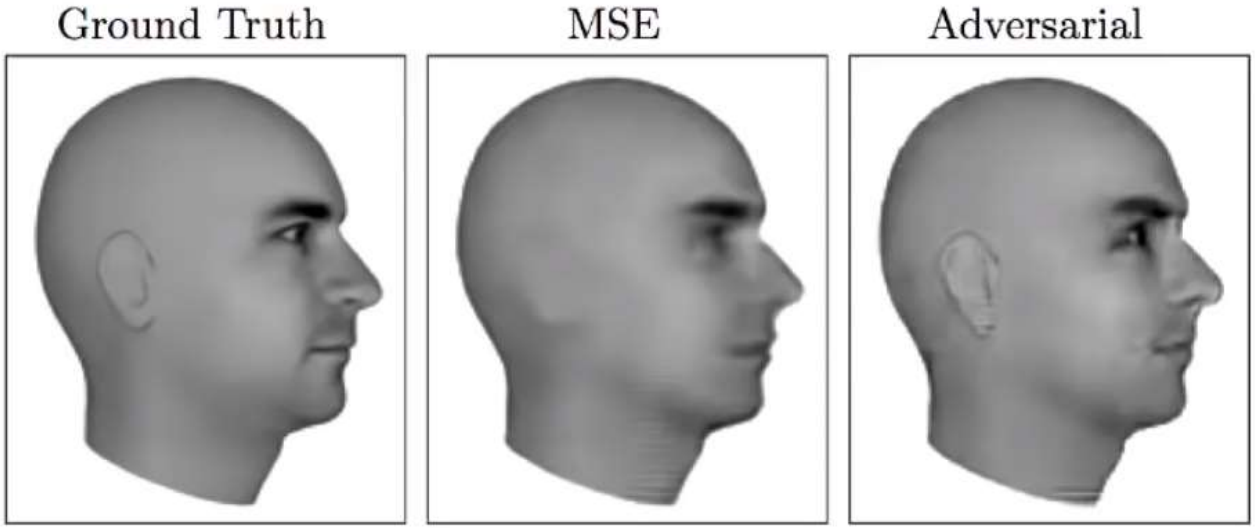
[IASI AI] GANS ALLOW MANY ANSWERS

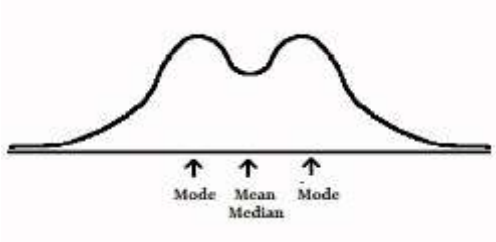


$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Adversarial error loss

NEXT VIDEO FRAME PREDICTION (2016)





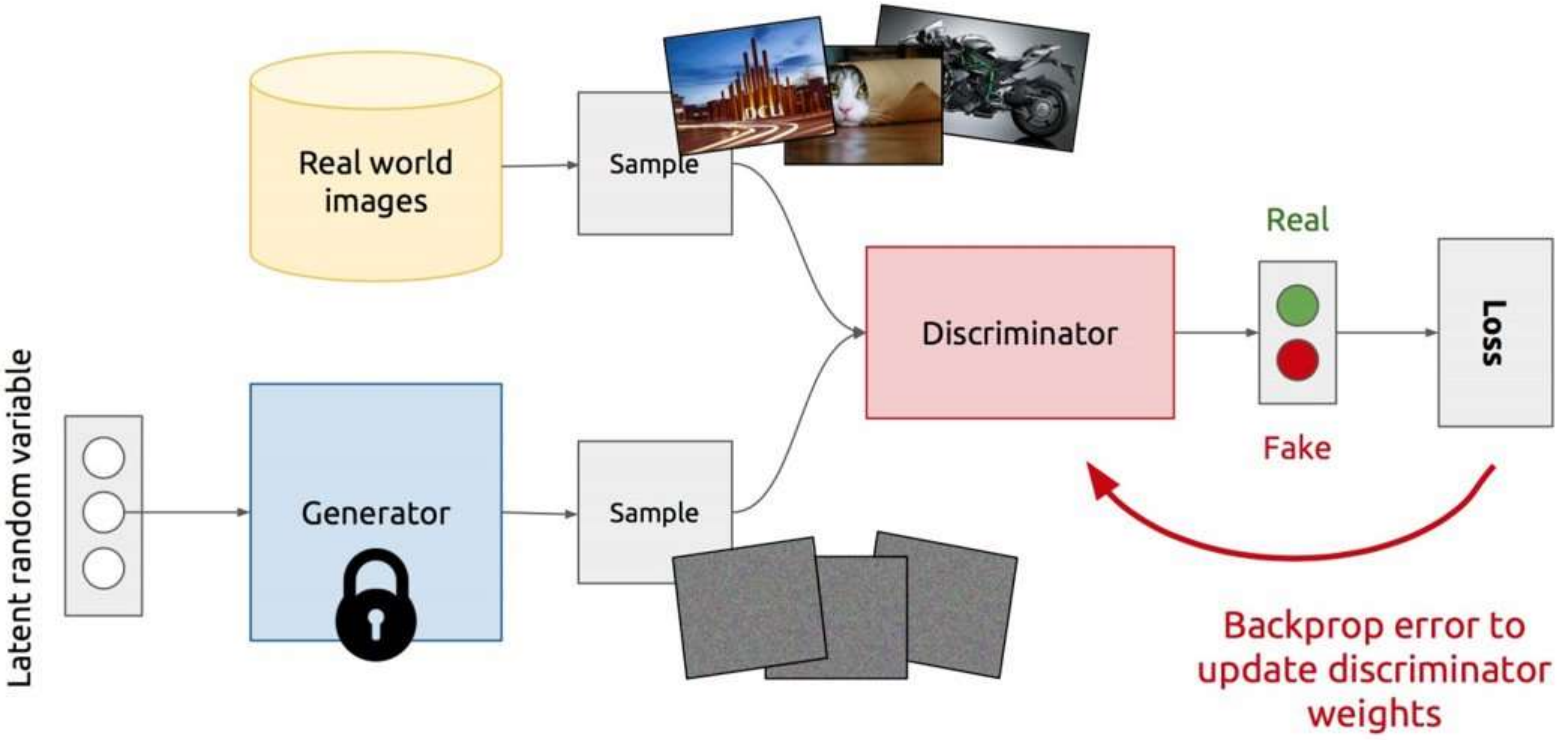
Adversarial loss learned by discriminator is much better vs MSE loss, it does not average between possible futures. Discriminator is being trained to weed out unrealistic examples which include blurring because of MSE Loss

If the probability distribution for an output pixel has equally likely modes v1 and v, then the v_avg = (v1 + v2)/2 minimizes the l2 loss over the data even id v_avg has very low probability

Code

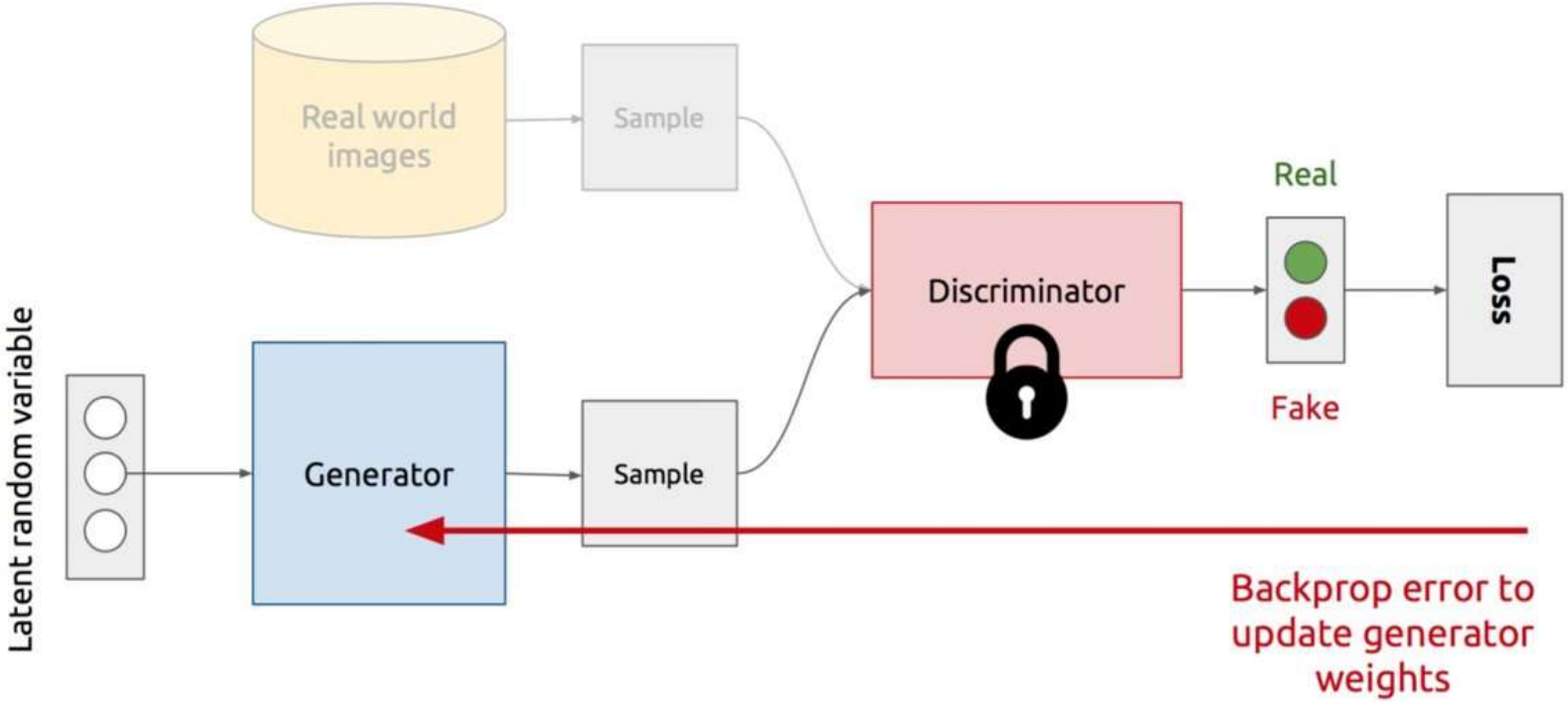
TRAINING DISCRIMINATOR

1. Fix generator weights, draw samples from both real world and generated images
2. Train discriminator to distinguish between real world and generated images



TRAINING GENERATOR

1. Fix discriminator weights
2. Sample from generator
3. Backprop error through discriminator to update generator weights



GENERAL GAN LOSS FUNCTION FORMULATION

The goal of the G is to minimize this loss whereas the D tries to maximize it.

The training objective for the two players can be described by an objective function of the form:

$$L = f(D(x)) - f(D(G(z)))$$ for some real-valued scoring function f.

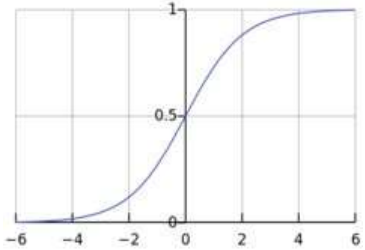
Our goal when training GANs is to find a Nash-equilibrium, i.e. a parameter assignment where neither the D nor the G can improve their utilities unilaterally.

The common choice is (negated) logistic loss which leads to original GAN paper:

$f(t) = -\log(1 + \exp(-t))$ = log(sigmoid), where $f'(t) \neq 0$ for all $t \in \mathbb{R}$.

, in this case the discriminator outputs the real-valued “logits” and the loss function would implicitly scale this to a probability

Sigmoid: $\frac{1}{1+e^{-x}}$



Log(1/x) = - log (x) => f(t) = log (sigmoid)

Log rules

Gradient descent GAN optimization is locally stable

ORIGINAL GAN LOSS FUNCTION FORMULATION

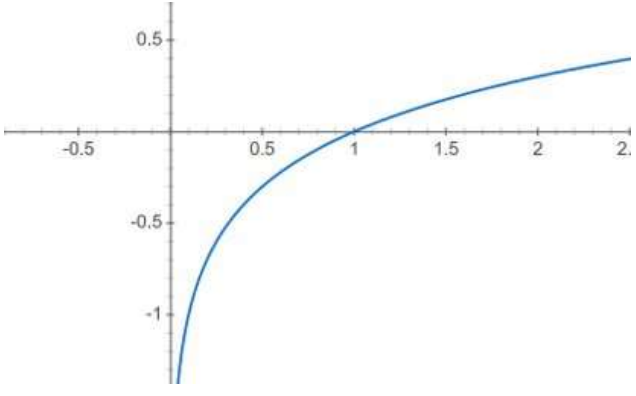
It is formulated as a minimax (zero-sum non-cooperative) game, where:

- The Discriminator is trying to maximize its reward V(D,G)
- The Generator is trying to minimize Discriminator's reward (or maximize its loss)

$$V(D,G) = \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim q(z)}[\log(1 - D(G(z)))] \qquad \min_G \max_D V(D,G)$$

The GAN formula is derived from log loss used for binary classification (or binary cross entropy function) (Why?)

Why log is used? It makes eq. easier to differentiate and is monotonically increasing function which means finding min or max is the same:



Log(xy) = log x + log y

Understanding Generative Adversarial Networks

GAN TRAINING ALGORITHM

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

Discriminator updates

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

Generator updates

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Criticizing is easy -> In practice we never train D to convergence. In fact, usually the discriminator is too strong, and we need to alternate gradient updates between D and G to get reasonable generator updates
Inverting D,G loops -> overfitting

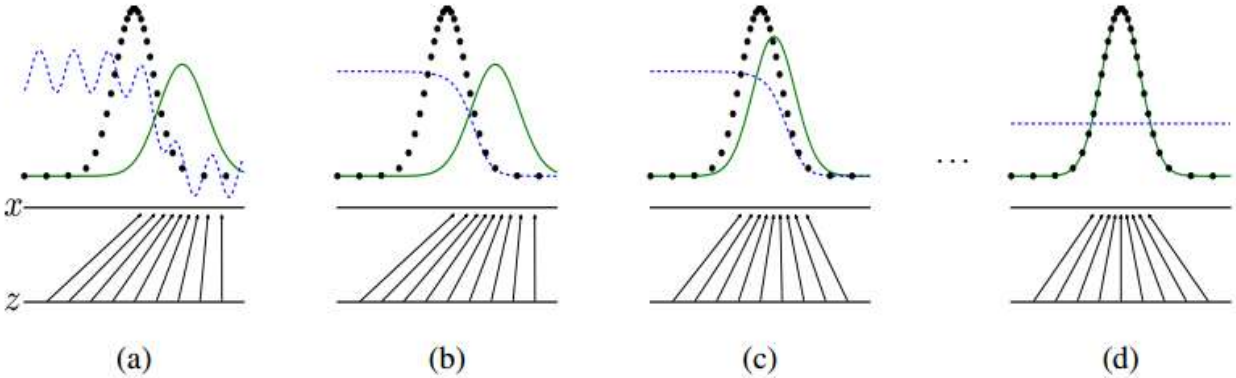
[IASI AI] WHAT IS DISCRIMINATOR LEARNING ?

The Nash equilibrium of this game is achieved when D is optimal and true and fake probability density functions are equal

D optimal $= \frac{p_r(x)}{p_r(x)+p_g(x)} \in [0,1].$

- $P_{data}(x) = P_{gen}(x) \ \forall x$
- $D(x) = \frac{1}{2} \ \forall x$

The math is here. One issue: derivative of D optimal can be unbounded or even incomputable => we need to regulate it (paper)



1D example of GAN

- black dotted = data distrib.
- green = model distrib.
- dotted blue = D

Understanding Generative Adversarial Network

From GAN to WGAN

As we see above **D tries to estimate the ratio between real and fake probability density functions.**
Original GAN uses learns JS divergence between true and fake distributions.

VANISHING GRADIENT PROBLEM IF D IS TOO GOOD + SOLUTION

Vanishing gradient strikes back again...

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

$$\nabla_{\theta_G} V(D,G) = \nabla_{\theta_G} \mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

- $\nabla_a \log\left(1-\sigma(a)\right) = \frac{-\nabla_a \sigma(a)}{1-\sigma(a)} = \frac{-\sigma(a)\left(1-\sigma(a)\right)}{1-\sigma(a)} = -\sigma(a) = -D\left(G(z)\right)$
- Gradient goes to 0 if D is confident, i.e. $D\left(G(z)\right) \to 0$

Math Formulas:
Deriv (F o G) = Deriv F (G) * Deriv G
Deriv Log2(x) = 1 / (*x* ln(2))
Deriv σ(x) = σ(x)·(1−σ(x))

- Minimize $-\mathbb{E}_{z\sim q(z)}\left[\log D\left(G(z)\right)\right]$ for **Generator** instead (keep Discriminator as it is)

Understanding Generative Adversarial Networks

DRAWBACKS OF USING GANS – TRAINING IS HARD

Non-convergence

The model parameters oscillate, destabilize and never converge,

“Oscillation” = can train for a very long time, generating very many different categories of samples, without clearly generating better samples

Highly sensitive to the hyperparameter selections.

Mode collapse

The generator collapses which produces limited varieties of samples, (most severe form of non-convergence)

Unbalance between the generator and discriminator causing overfitting .

GANS NON-CONVERGENCE PROBLEM - EXERCISE

GAN is based on the zero-sum non-cooperative game. In short, if one wins the other loses.
Since both sides want to undermine the others, a **Nash equilibrium** happens when one player will not change its action regardless of what the opponent may do.

Consider two player A and B which control the value Y and X respectively.
Player A wants to maximize the value XY while **B** wants to minimize it. Each make a move at a time.

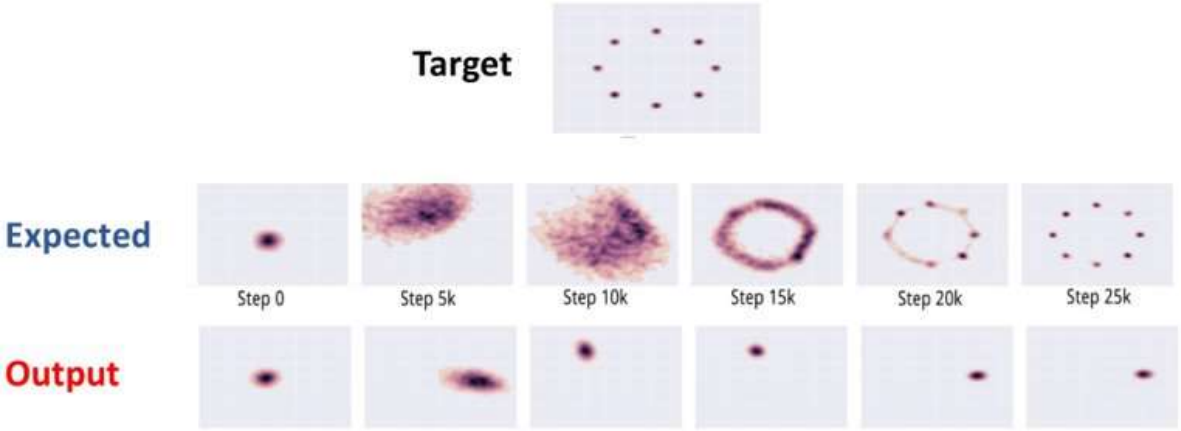
$$\min_{B} \max_{A} V(D, G) = xy$$

• State 1:	x > 0	y > 0	V > 0	Increase y	Decrease x
• State 2:	x < 0	y > 0	V < 0	Decrease y	Decrease x
• State 3:	x < 0	y < 0	V > 0	Decrease y	Increase x
• State 4 :	x > 0	y < 0	V < 0	Increase y	Increase x
• State 5:	x > 0	y > 0	V > 0	Increase y	Decrease x

== State 1

MODE-COLLAPSE PROBLEM

Generator fails to output diverse samples:



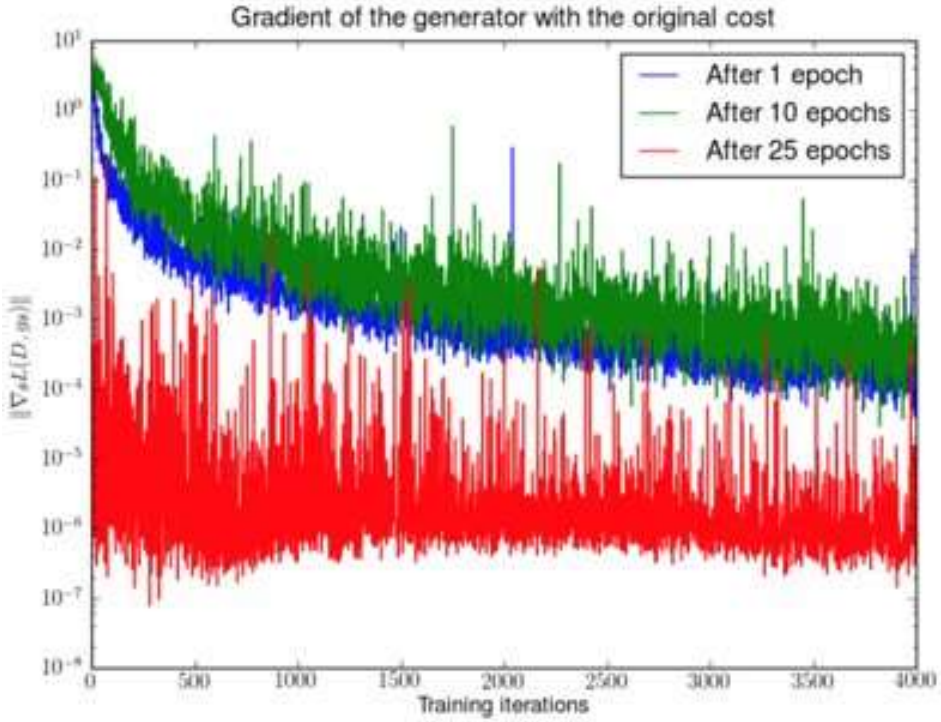
D in inner loop: convergence to correct distribution / G in inner loop: place all mass on most likely point:

$$\min_G \max_D V(G,D) \neq \max_D \min_G V(G,D) \qquad x^* = argmax_x D(x)$$

Let's consider one extreme case where ***G*** is trained extensively without updates to ***D***.
The generated images will converge to find the optimal image ***x**** that fool ***D*** the most, the most realistic image from the discriminator perspective. In this extreme, ***x**** will be independent of ***z***. => This is bad news.
The mode collapses to a **single point**. … => networks are overfitted to exploit the short-term opponent weakness, a never-ending cat-and-mouse game

VANISHING (SATURATING) GRADIENTS IN G

For original cost function the discriminator gets too successful that the generator gradient vanishes and learns nothing



GAN - What is wrong with the GAN cost function?
Why it is so hard to train Generative Adversarial Networks!

From GAN to WGAN

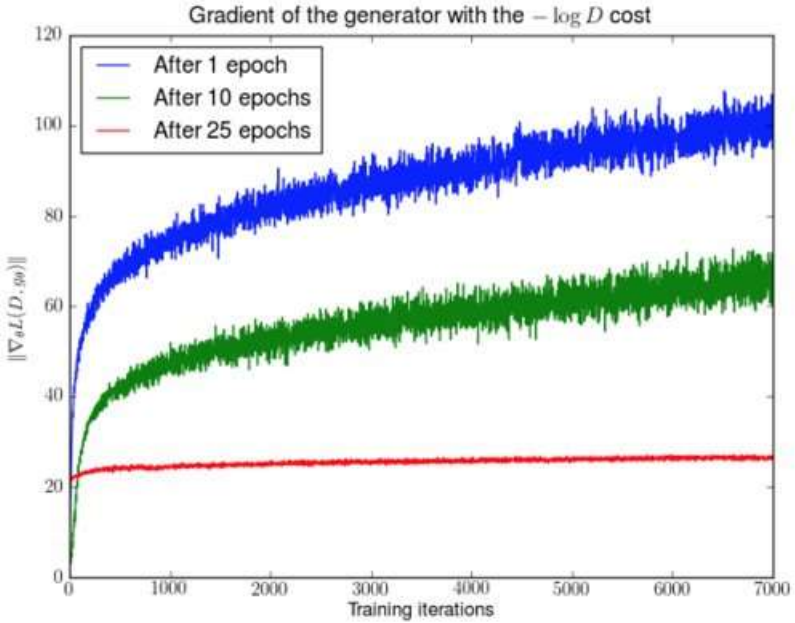
UNSTABLE GRADIENTS FOR G

And the alternative cost function has fluctuating gradients that cause instability to the models :

$$\nabla_{\theta_g} - \log D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)$$

unstable with large variance of gradients

Alternative proposal



Here, the generator is fixed and the experiment optimizes the discriminator again. The diagram below plots the gradient changes during this training. As shown, not only the gradient goes up but it fluctuates more. All these lead to unstable models.

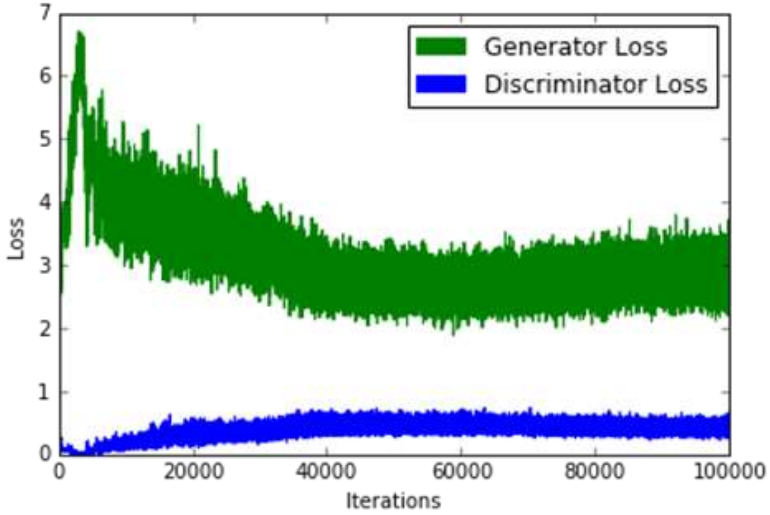
Other issue: derivative of D optimal can be unbounded or even incomputable => we need to regulate it (paper)
The math is here.

GAN - What is wrong with the GAN cost function?
Why it is so hard to train Generative Adversarial Networks!
From GAN to WGAN

ISSUES WITH QUALITY AND METRICS

Lack of a proper evaluation metric:

- No good sign to tell when to stop training. Since the Generator loss improves when the Discriminator loss degrades (and vice-versa), we can't judge convergence based on the value of the loss function.
- The GAN objective function explains how well the G or D is performing with respect to its opponent. It does not however represent the quality or the diversity of the output.
- No good indicator to compare the performance of multiple models. It is difficult to quantitatively tell when the generator produces high quality samples



Advances in Generative Adversarial Networks

GAN ADVANCES (2014 - 2016)

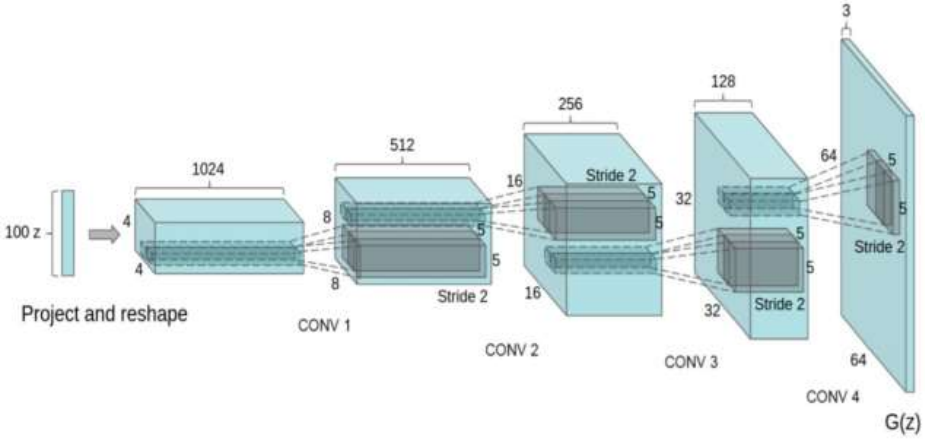
Use Depth and Convolution for Harder Tasks (DCGAN)

Key ideas:

- Replace FC hidden layers with Convolutions
- Use Batch Normalization after each layer

Inside Generator

- Transposed convolutions (for up-sampling)
- Use (Leaky) ReLU for hidden layers
- Use Tanh for the output layer
- Get rid of the pooling layers



Add supervision with labels (Class-conditional generation)

D should output also labels for real data / Add labels to D and G input => Empirically generates much better samples (Semi-supervised training = Used standalone D will be better regularized classifier and needs very few labeled data to train)

Minibatch discrimination

D should decide also based on similarity in a batch of data to discourage mode collapse in G.

Improved Techniques for Training GANs

Unrolled GANs (how to reduce mode collapse)

Feature Matching (Perceptual loss)

Feature matching expands the goal from beating the opponent to matching features in real images. Here is the new objective function:

$$||\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\mathbf{f}(G(\boldsymbol{z}))||_2^2$$

***Where f(x)* is the feature vector extracted from an immediate layer by the discriminator output.**

- useful for image to image translation

More specific formulation:

a. take the moments for some features for a “real” minibatch

b. take the moments for same features for a “fake” minibatch

c. minimize mean absolute error between the two

Ways to improve GAN performance

Tips and tricks to make GANs work

GAN ADVANCES (AFTER 2016)

1-Lip: $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$.

Various types of network regularization & normalization

It is useful to improve training stability and network generalization power by limiting gradients and weight values:

- 1 gradient penalty regularization:

$$\lambda \underset{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}{\mathbb{E}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]$$

- 0 gradient penalty regularization

$$\mathcal{L}_{0-GP} = \mathcal{L} - \lambda \mathbb{E}_{\tilde{\boldsymbol{x}}} [\|(\nabla D)_{\tilde{\boldsymbol{x}}}\|^2]$$

- Orthogonal weight matrix regularization

$$\mathcal{L}_{ortho} = \Sigma(|WW^T - I|)$$

- Spectral weight matrix normalization

$$\bar{W}_{SN}(W) := \frac{W}{\sigma(W)} \qquad \sigma(A) := max_{h:h \neq 0} \frac{\|Ah\|_2}{\|h\|_2}$$

- Instance normalization
- Pixel normalization

[IASI AI] GAN ADVANCES (AFTER 2016) - ALTERNATE LOSS FUNCTIONS

Use Earth-Mover (EM) distance / Wasserstein Metric (WGAN, WGAN-GP)

It is a better metric for D to learn to measure distance between true/fake distributions (instead of classic KL-Divergence or JS-Divergence)

Hinge loss

$$V_D(\hat{G}, D) = \mathop{\mathrm{E}}_{\boldsymbol{x}\sim q_{\text{data}}(\boldsymbol{x})}\left[\min\left(0, -1 + D(\boldsymbol{x})\right)\right] + \mathop{\mathrm{E}}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\min\left(0, -1 - D\left(\hat{G}(\boldsymbol{z})\right)\right)\right]$$

$$V_G(G, \hat{D}) = -\mathop{\mathrm{E}}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\hat{D}\left(G(\boldsymbol{z})\right)\right],$$

Note: don't punish too harsh beginners

Other losses

	f	g	h
minimax (Goodfellow et al., 2014)	$-\log(1+e^{-y})$	$-y-\log(1+e^{-y})$	$-y-\log(1+e^{-y})$
nonsaturating (Goodfellow et al., 2014)	$-\log(1+e^{-y})$	$-y-\log(1+e^{-y})$	$\log(1+e^{-y})$
Wasserstein (Arjovsky et al., 2017)	y	$-y$	$-y$
least squares (Mao et al., 2017)	$-(y-1)^2$	$-y^2$	$(y-1)^2$
hinge (Lim & Ye, 2017; Tran et al., 2017)	$\min(0, y-1)$	$\min(0, -y-1)$	$-y$

$$\max_D \mathbb{E}_{\mathbf{x}\sim p_d}[f(D(\mathbf{x}))] + \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g}[g(D(\tilde{\mathbf{x}}))]$$

$$\min_G \mathbb{E}_{\tilde{\mathbf{x}}\sim p_g}[h(D(\tilde{\mathbf{x}}))],$$

Towards a Deeper Understanding of Adversarial Losses
Are GANs Created Equal? A Large-Scale Study

[IASI AI]

ORTHOGONAL MATRICES

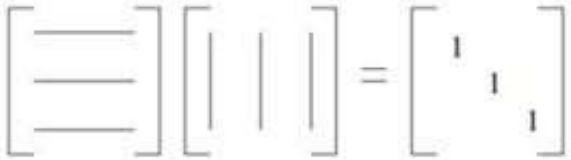


Figure 1.1: Orthogonal Matrices.

$$Q^{\mathrm{T}}Q = QQ^{\mathrm{T}} = I,$$ where I is the identity matrix

A matrix *Q* is orthogonal if its transpose is equal to its inverse: $Q^{\mathrm{T}} = Q^{-1}$

Orthogonal matrices preserve length of a vector: $\|Qx\|_2 = \|x\|_2$

Orthogonal matrices preserve 2-norm and Frobenius norm of a matrix:

$$\|QA\|_2 = \|A\|_2 \qquad \|QA\|_F = \|A\|_F$$

MIT Orthogonal Matrices Course

MATRIX SPECTRAL NORM

Matrix norm gives the maximum gain or amplification of A.
Intuitively, you can think of it as the maximum 'scale', by which the matrix can 'stretch' a vector. (MathExch) (Stanford) (Khan academy – a matrix can transform a vector), Wolfram :

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

$$\|A\| = \sup_x \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|$$

Matrix norm equals also square root of max eigenvalue (largest sigural value) of matrix A:

$$\|A\| = \sqrt{\lambda_{\max}(A^{\mathsf{T}}A)}$$

Proof:
Expl: Max T(A)*A*x = max eigenvalue * x

$$\max_{x \neq 0} \frac{\|Ax\|^2}{\|x\|^2} = \max_{x \neq 0} \frac{x^{\mathsf{T}}A^{\mathsf{T}}Ax}{\|x\|^2} = \lambda_{\max}(A^{\mathsf{T}}A)$$

(Why)

Stanford course

Frobenius norm (used for weight decay regularization):

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\operatorname{trace}(A^*A)} = \sqrt{\sum_{i=1}^{\min\{m,n\}}\sigma_i^2(A)},$$

where $\sigma_i(A)$ are the singular values of A. Recall that the trace function returns the sum of diagonal entries of a square matrix.

Wiki

GAN ADVANCES (AFTER 2016)

Two Time-Scale Update Rule

One to one generator/critic iterations and higher critic learning rate – more efficient than doing multiple D iterations

Self-Attention (SAGAN)

Helpful for DCGANs by modeling long distance spatial dependencies

Relativistic GAN

let the discriminator predict relative realness instead of the absolute value

Progressive Growing of GANS (ProGAN)

Add new layers as size of the input image grows during training (curriculum learning)

Scale up the network size (BigGAN)

Noise Truncation Trick (BigGAN)

Truncate input noise space after training to balance network creativity (output diversity) with output quality

Hierarchical Latent spaces (BigGAN & StyleGAN)

Noise is passed through a MLP and then split and fed to each layer to control its activation statistics (Batch norm)

Dive head first into advanced GANs: exploring self-attention and spectral norm

[IASI AI]

SAGAN (SELF-ATTENTION GAN)

Problem:

CGAN has problem with classes with complex geometry. The CGAN was able to generate the texture of furs of dog but was unable to generate distinct legs.

Cause:
Convolution is a local operation, limited receptive field. Unable to model long distance spatial relationships.

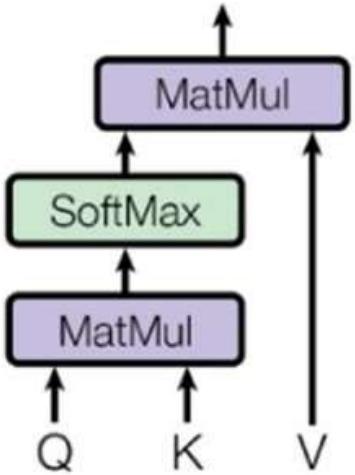
Solution like enlarging filter size or shrinking filter size and add more layers are computational expensive.

Solution: Self-Attention GANs

It helps create a balance between efficiency and long-range dependencies(= large receptive fields) by utilizing the famous mechanism from NLP called attention.

Attention is when the *query* and *key* decide how much the *value* can speak to the outer world. In self-attention, the query, the key and, the value are all same.

Ex: The region for left ear queries region for the right ear to decide if a left ear is really in this region and is the right output.



SELF-ATTENTION MAPS

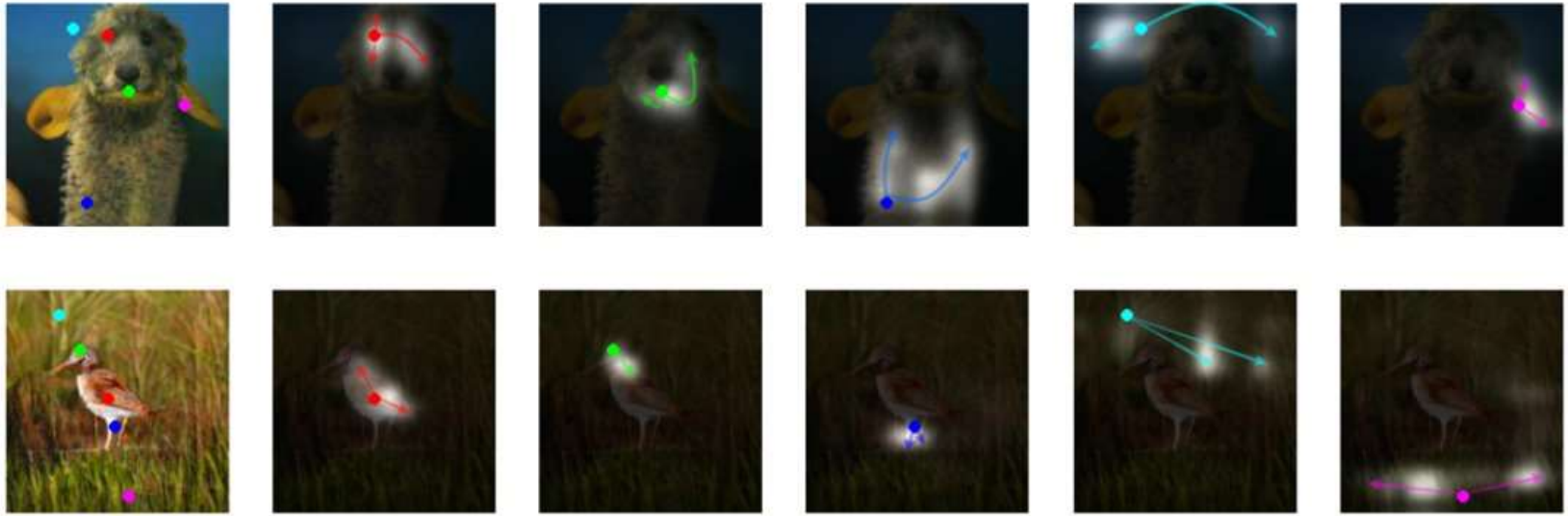


Figure 1: The proposed SAGAN generates images by leveraging complementary features in distant portions of the image rather than local regions of fixed shape to generate consistent objects/scenarios. In each row, the first image shows five representative query locations with color coded dots. The other five images are attention maps for those query locations, with corresponding color coded arrows summarizing the most-attended regions.

[IASI AI]

SELF-ATTENTION MECHANISM

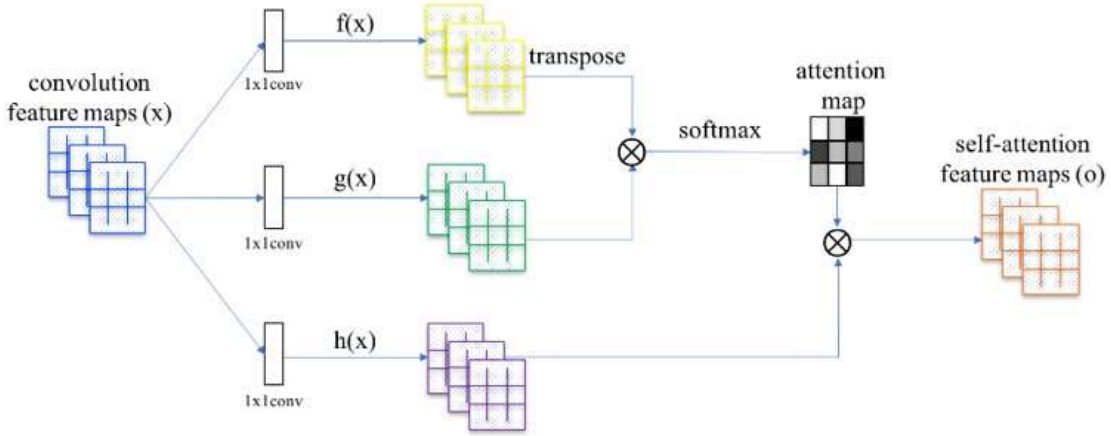


Figure 2: The proposed self-attention mechanism. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

1) Initially we have (512 x 7x 7) where 512 is the number of channels and 7 is the spatial dimension
   f and *g* have 64 those filters , *h* has 512 of those filters
2) Query = (64 x 7 x 7), Key = (64 x 7 x 7), Value=(512 x 7 x 7)
3) flatten out last two dimensions => Q (64 x 49), K (64 x 49) and V (512 x 49) ,
4) do A = SoftMax(T(Q) * K) => A (49 x 49)
5) V * A has (512 x 49)
6) if *x* was the initial image and ***o*** is the self-attention output the final output is ***O=y*o+x*** , y = learnable parameter initially 0
This means that self-attention is learned to be used gradually.

[IASI AI] HOW TO MEASURE GAN PERFORMANCE - INCEPTION SCORE (IS)

How to measure quality?

We use an Inception-v3 network pre-trained on ImageNet to classify the generated images and predict *P(y | x),* the label of *x*. Ideally, the generator should generate images ***with meaningful objects***, so that the conditional label distribution P(y | x) is *low entropy.*

How to measure the diversity of images?

If the generated images are diverse, the data distribution for labels *y* should be uniform (high entropy)

Marginal Probability - probability of any single event occurring unconditioned on any other events. (Quora)

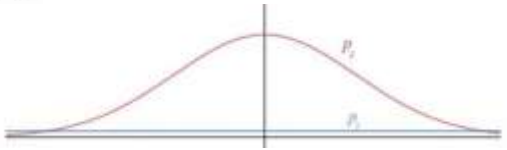
$$p(y) = \int_x p(y|x)p_g(x)$$

Intuitively the marginal probability of X is computed by examining the conditional probability of X given a particular value of Y, and then averaging this conditional probability over the distribution of all values of Y (Wiki)

To combine these two criteria, we compute their KL-divergence (entropy distance) and use the equation below to compute IS.

$$\text{IS} = \exp\left(\mathbb{E}_{x \sim p_g} D_{KL}(p(y|x)||p(y))\right)$$

Disadvantage: if it only generates one image per class. *p(y)* will still be uniform even though the diversity is low



How to measure GAN performance?

[IASI AI] HOW TO MEASURE GAN PERFORMANCE - FRÉCHET INCEPTION DISTANCE (FID)

The FID is supposed to improve on the IS by comparing the statistics of generated samples to real samples, instead of evaluating generated samples in a vacuum. (Heusel, Ramsauer, Unterthiner, Nessler, & Hochreiter, 2017)

We use the Inception network to extract features from an intermediate layer. Then we model the data distribution for these features using a multivariate Gaussian distribution with mean μ and covariance matrix Σ. The FID between the two multivariate Gaussians for the real images x and generated images g is computed as:

$$\text{FID} = ||\mu_r - \mu_g||^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

, where *Tr* (=trace matrix) sums up all the diagonal elements.

Lower FID values mean better image quality and diversity.

Advantage: If the model only generates one image per class, the FID distance will be high. So FID is a better measurement for image diversity.

(Overall Measures of Dispersion = Total Variance)

How to measure GAN performance?

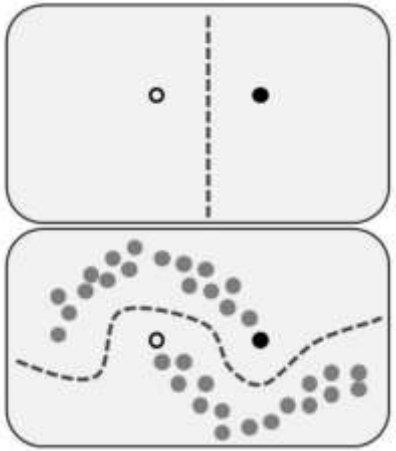
Fréchet Inception Distance (Code & FID for various GAN arhitectures scores here)

SEMI-SUPERVISED LEARNING WITH GANS

Why?

- Create a more diverse set of unlabeled data
- Get a better / more descriptive decision boundary
- Improve generalization when the amount of training samples is small

Influence of unlabeled data



- Continuity assumption
- Cluster assumption
- Manifold assumption

[IASI AI] GENERATIVE MODELING REVEALS A FACE

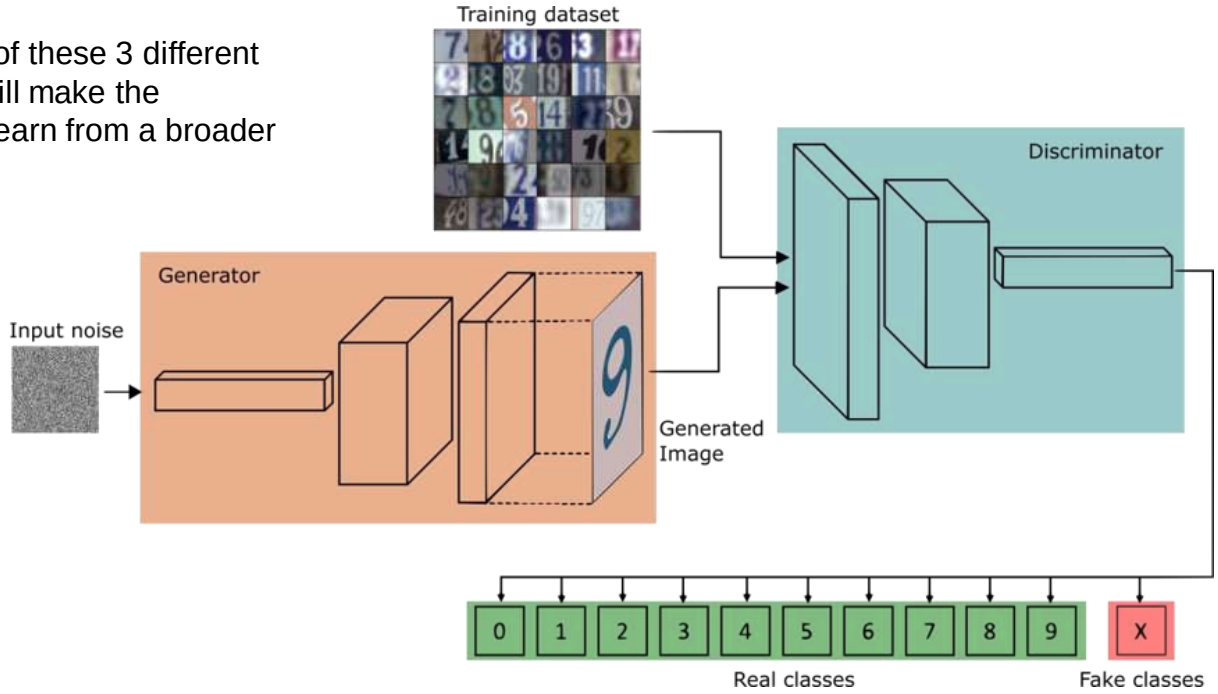
Semantic Image Inpainting with Deep Generative Models

Overcoming Limited Data with GANs

[IASI AI]

GAN IN A SEMI-SUPERVISED LEARNING ROLE

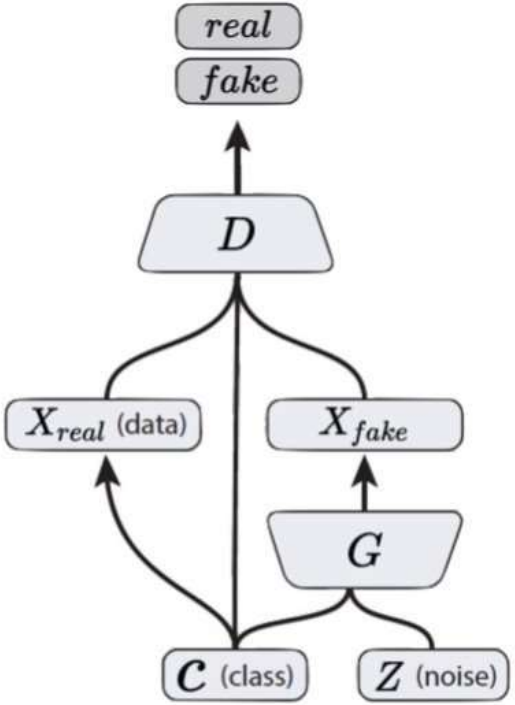
The combination of these 3 different sources of data will make the classifier able to learn from a broader perspective.



Semi-supervised learning with Generative Adversarial Networks (GANs)

[IASI AI] Famous GAN Types

- Conditional GAN
- InfoGAN
- SemiSupervised GAN
- Wasserstein GAN
- Pix2Pix (HD)
- CycleGAN
- RecycleGAN
- SaGAN
- ProGAN
- BigGAN
- StyleGAN
- InstaGAN



Conditional GAN
(Mirza & Osindero, 2014)

IMPLEMENTING A TOY GAN

- MNIST Generative Adversarial Model in Keras
- Generative Adversarial Networks (GANs) in 50 lines of code (PyTorch)
- Introductory guide to Generative Adversarial Networks (GANs) and their promise!
- Understanding and optimizing GANs (Going back to first principles)
- Keras-GAN Implementations

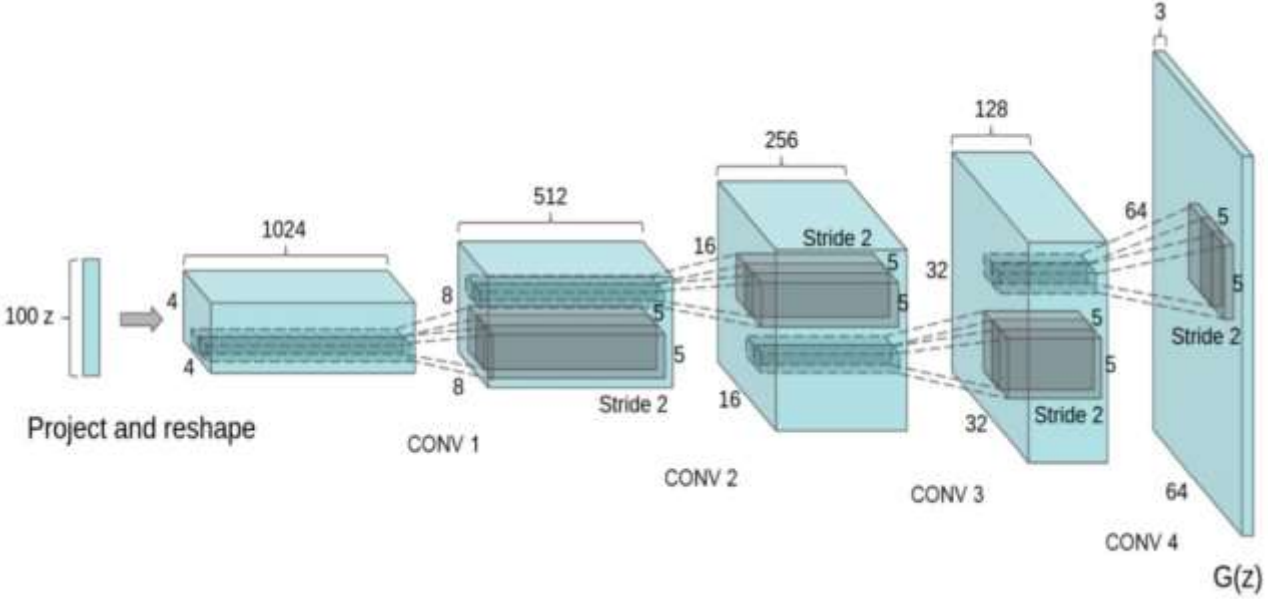
[IASI AI]

DEEP CONVOLUTIONAL GANS (DCGAN)

A major improvement in image generation occured in 2016, when Radford et al published Unsupervised Representation Learning With DCGANs. DCGANs got rid of the pooling layers used in some CNNs and relied on convolutions in D and transpose convolutions in G to change the representation size. Most layers were followed by a batch normalization and a leaky ReLU activation.

Tutorial with Code

DCGAN ARCHITECTURE



Key ideas:

- Replace FC hidden layers with Convolutions
- Use Batch Normalization after each layer

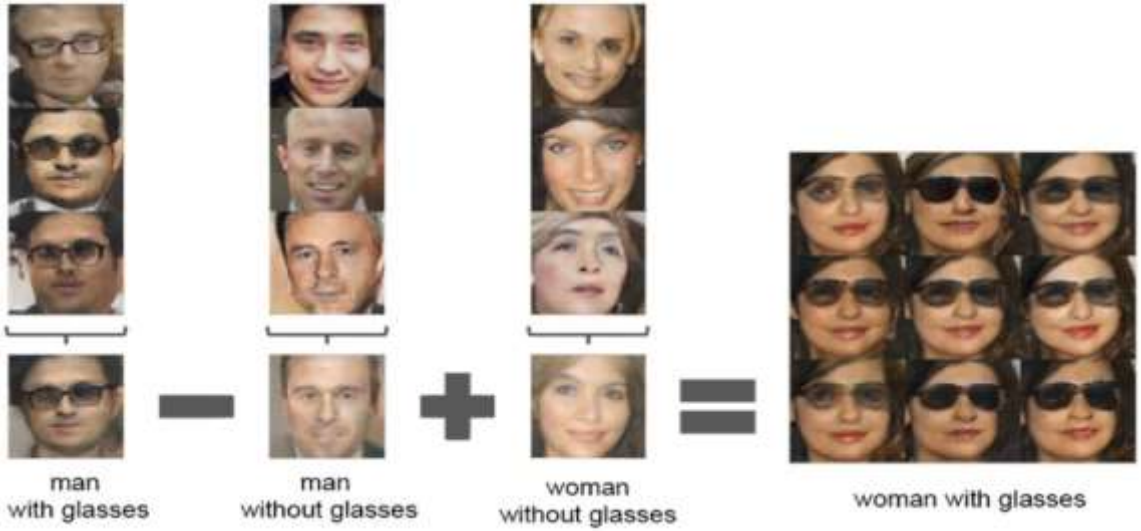
Inside Generator

- Transposed convolutions (for up-sampling)
- Use (Leaky) ReLU for hidden layers
- Use Tanh for the output layer
- Get rid of the pooling layers

Problems: Yet even DCGANs could only create images of a certain size. The higher the resolution of an image, the easier it becomes for the discriminator to tell the “real” images from the “fakes”. This makes mode collapse more likely. While synthesizing 32x32 or even 128x128 images became routine tutorial material, generating images at resolutions above 512x512 remained challenging in practice.

DCGAN- LATENT VECTORS CAPTURE INTERESTING PATTERNS...

DCGANs demonstrated that GANs can learn a distributed representation that disentangles the concept of gender from the concept of wearing glasses.



Note: Similar vector space arithmetic like with word embeddings

Unsupervised representation learning with DCGANs

CONDITIONAL GANS - CGAN

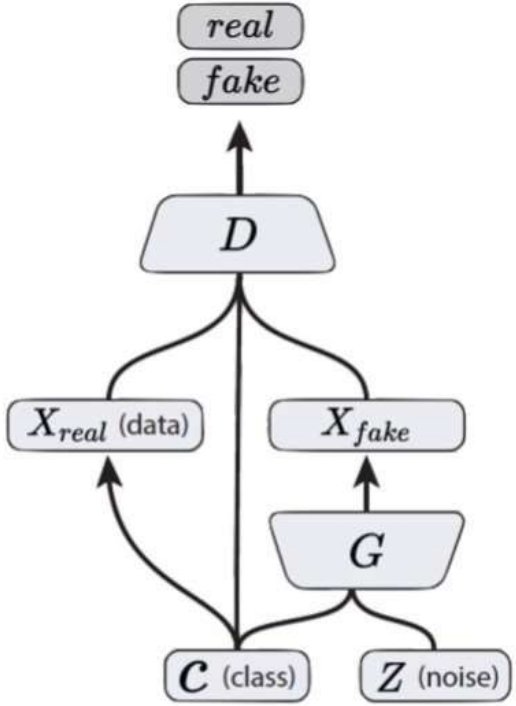
GAN can generate random digits that look like the ones from the MNIST data set. But what if we want to generate specific digits?

MNIST digits generated conditioned on their class label.

Conditional generative adversarial nets (2014 Paper)

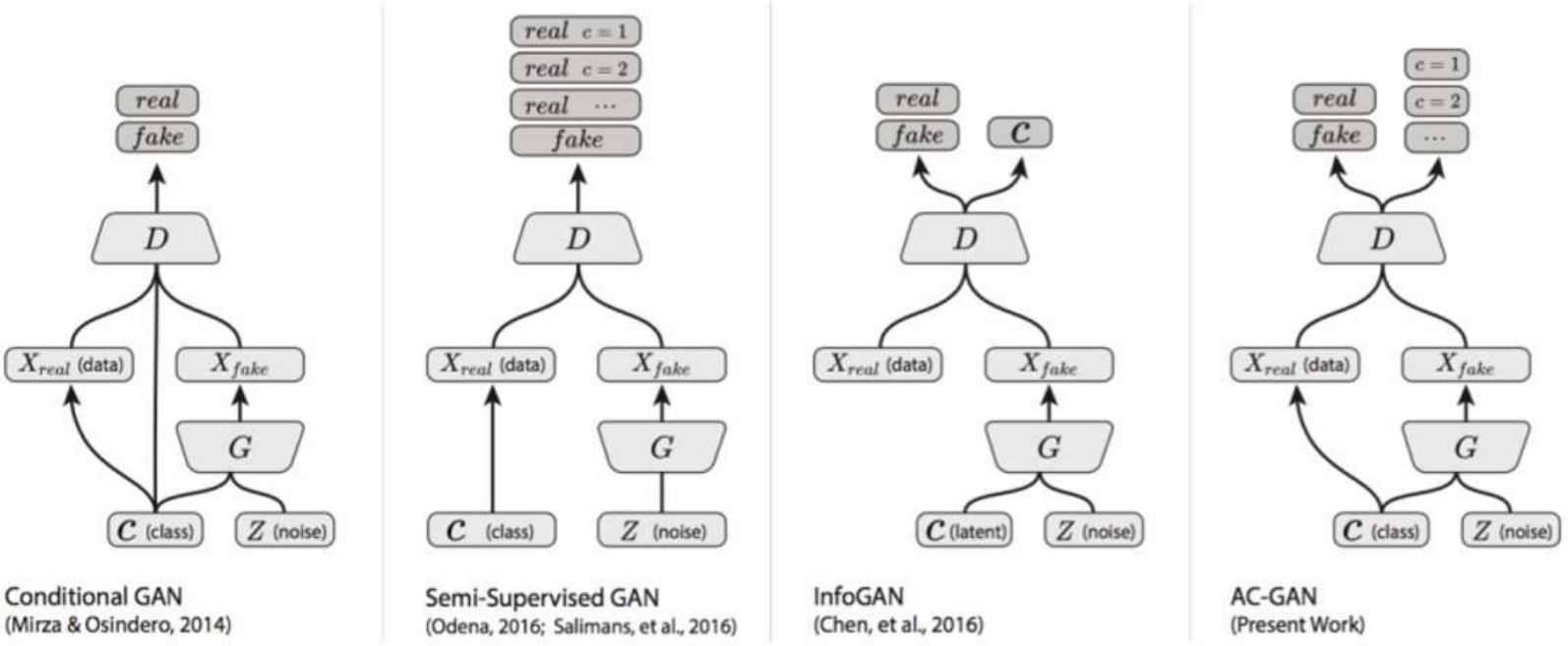
CONDITIONAL GANS – CGAN ACHITECTURE

- Simple modification to the original GAN framework that conditions the model on additional information for better multi-modal learning.
- Lends to many practical applications of GANs when we have explicit supervision available.



Conditional GAN
(Mirza & Osindero, 2014)

VARIATIONS OF CONDITIONAL GAN



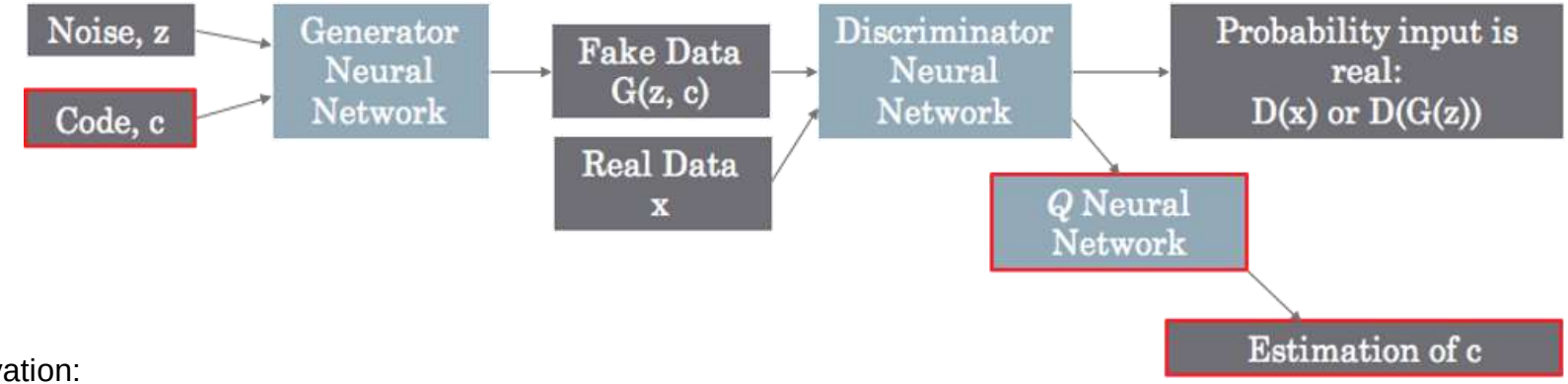
Unsupervised Image-to-Image Translation with Generative Adversarial Networks

Mutual information (additional loss term used by InfoGAN) ,

InfoGAN: unsupervised conditional GAN in TensorFlow and Pytorch

INFOGAN - ARCHITECTURE

New components outlined in red:



Motivation:
there are *structures* in the noise vectors that have meaningful and consistent effects on the generator output … but there's no intuition about how to modify it to get a desired effect => InfoGAN tries to solve this problem and provide a *disentangled representation*

$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c))$$

$$I(X;Y) = H(X) - H(X|Y).$$

$$L_I(G,Q) = E_{c\sim P(c), x\sim G(z,c)}[\log Q(c|x)] + H(c)$$

Add new term = mutual information between the latent code *c* and the generator output *G(z,c)*
Output of *Q* is not the code value itself, but instead the statistics of the distribution you chose to model the code c

InfoGAN - Generative Adversarial Networks
Mutual information (additional loss term used by InfoGAN) ,
InfoGAN: unsupervised conditional GAN in TensorFlow and Pytorch

[IASI AI]

EVOLUTION OF VARIOUS DISCRIMINATOR MODELS FOR CGAN

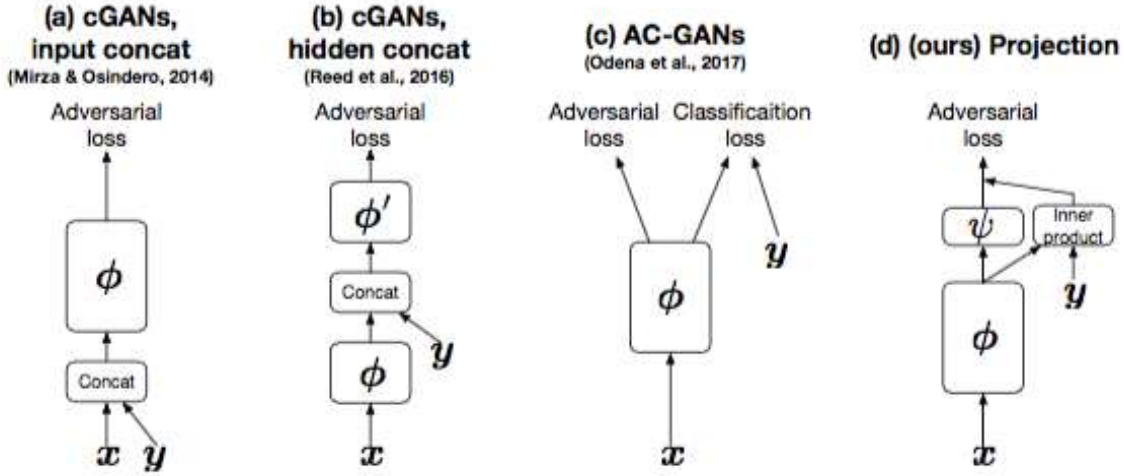


Figure 1: Discriminator models for conditional GANs

a) cGANs
input in the discriminator is changed to x plus y (ground truth), so that the discriminator can be trained better and more stable and simpler.

b) cGANs hidden concat
Found that the results of concat in the hidden layer and y will be better.

c) AC-GANs (D with Auxiliary classifier)
He not only has adversarial loss, but also outputs a classify result at the end, so the result is better (two loss eq).

d) Projection GAN
This method achieves better results than using concat alone.

cGANs with projection

[IASI AI]

GANS FOR SIMULATED TRAINING DATA

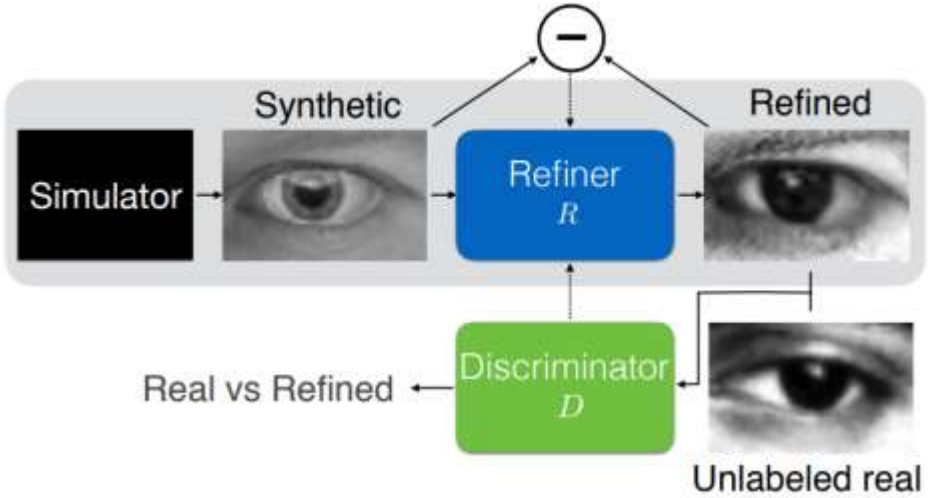


Figure 2. Overview of SimGAN. We refine the output of the simulator with a refiner neural network, R, that minimizes the combination of a local adversarial loss and a 'self-regularization' term. The adversarial loss 'fools' a discriminator network, D, that classifies an image as real or refined. The self-regularization term minimizes the image difference between the synthetic and the refined images. The refiner network and the discriminator network are updated alternately.

Simulated + Unsupervised learning. The task is to learn a model that improves the realism of synthetic images from a simulator using unlabeled real data, while preserving the annotation information.

Improving the Realism of Synthetic Images

Learning from Simulated and Unsupervised Images through Adversarial Training

[IASI AI]

SEMANTIC IMAGE INPAINTING - ARHITECTURE

Back-propagation to the input data is employed in our approach to find the encoding which is close to the provided but corrupted image.

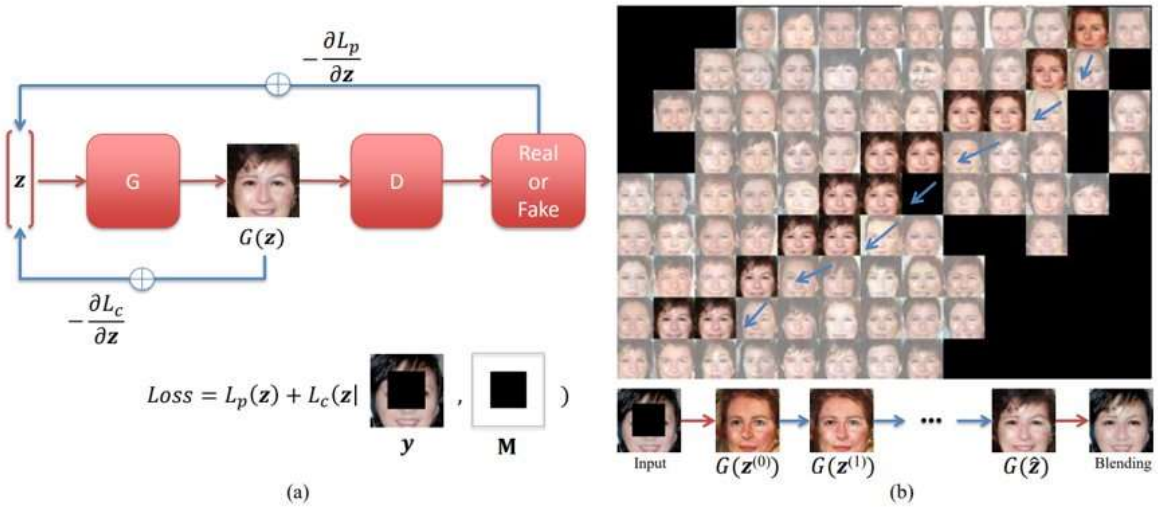


Figure 3. The proposed framework for inpainting. (a) Given a GAN model trained on real images, we iteratively update $\mathbf{z}$ to find the closest mapping on the latent image manifold, based on the desinged loss functions. (b) Manifold traversing when iteratively updating $\mathbf{z}$ using back-propagation. $\mathbf{z}^{(0)}$ is random initialed; $\mathbf{z}^{(k)}$ denotes the result in k-th iteration; and $\hat{\mathbf{z}}$ denotes the final solution.

$$\hat{\mathbf{z}} = \arg\min_{\mathbf{z}} \{\mathcal{L}_c(\mathbf{z}|\mathbf{y}, \mathbf{M}) + \mathcal{L}_p(\mathbf{z})\}$$

Lc = context loss, which constrains the generated image given the input corrupted image y and the hole mask M;

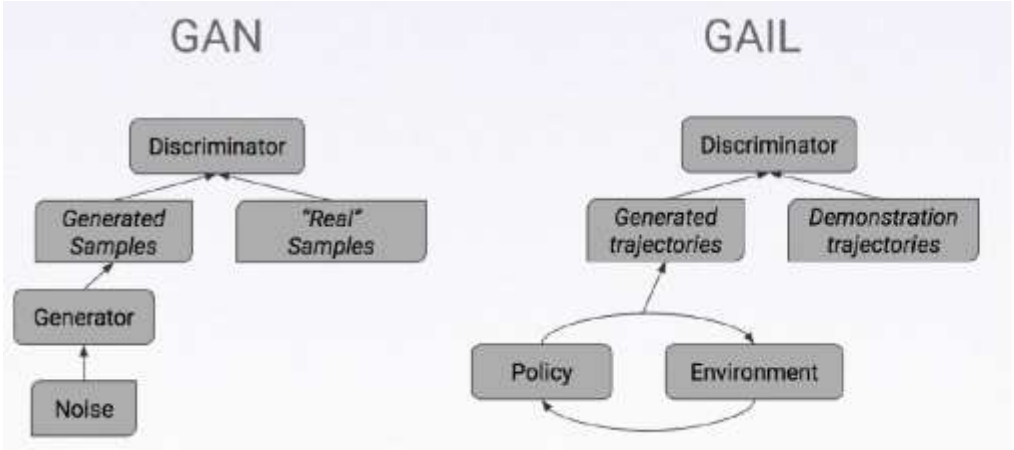
Lp = prior loss, which penalizes unrealistic images

Semantic Image Inpainting with Deep Generative Models

[IASI AI]

GANS FOR IMITATION LEARNING

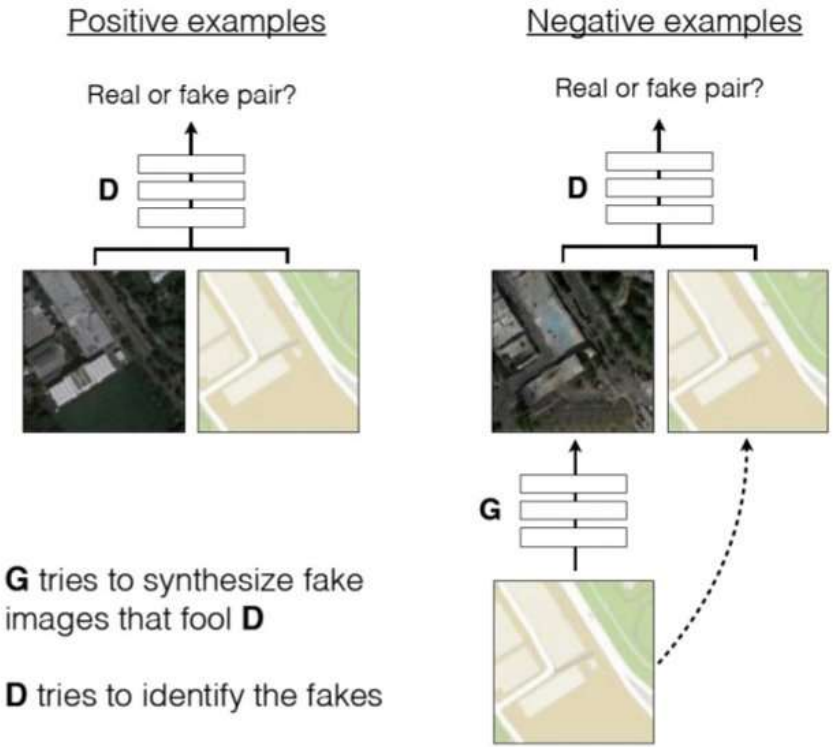
- Use a separate network (discriminator) to “learn” what is realistic
- Adversarial imitation learning: RL Reward comes from a discriminator



Learning human behaviors from motion capture by adversarial imitation

[IASI AI]

IMAGE-TO-IMAGE TRANSLATION WITH PIX2PIX

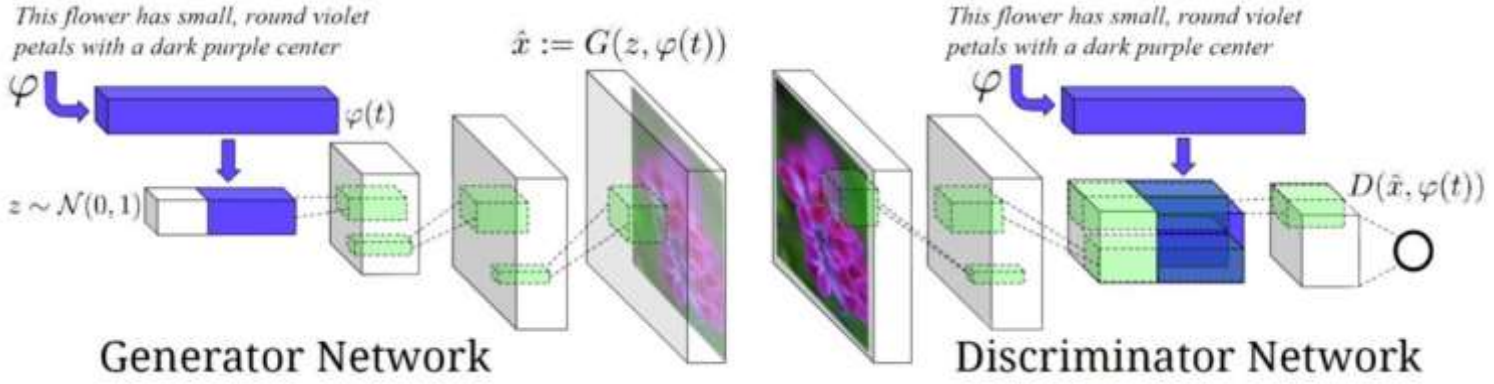


- Architecture: DCGAN (Deep convolutional generative adversarial networks) based architecture
- Training is conditioned on the images from the source domain.
- Conditional GANs provide an effective way to handle many complex domains without worrying about designing structured loss functions explicitly.

Image-to-image translation with conditional adversarial networks (Paper)

[IASI AI]

TEXT-TO-IMAGE SYNTHESIS

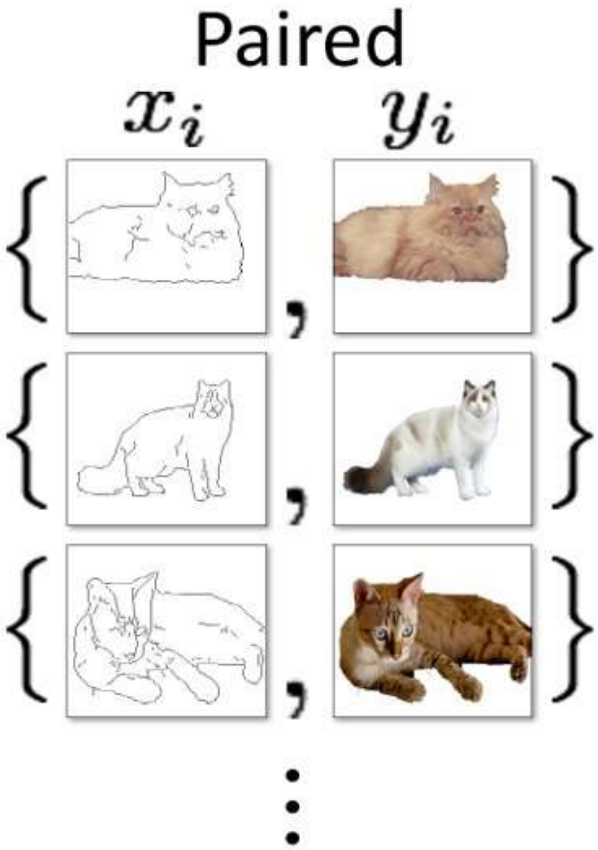


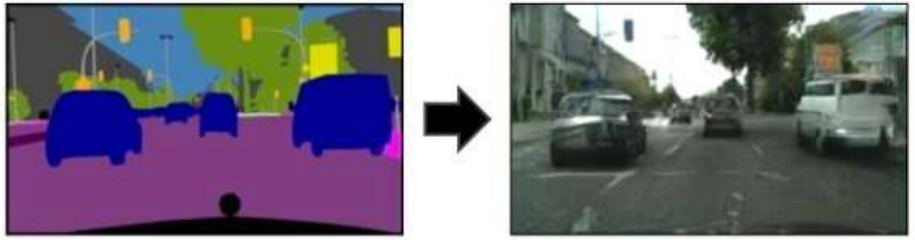
Positive Example: Real Image, Right Text

Negative Examples: Real Image, Wrong Text or Fake Image, Right Text

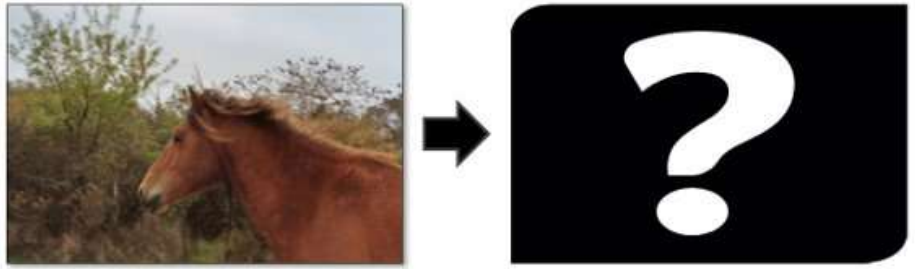
Generative adversarial text to image synthesis (2016 paper)

PAIRED IMAGE-TO-IMAGE TRANSLATION UNPRACTICALITY





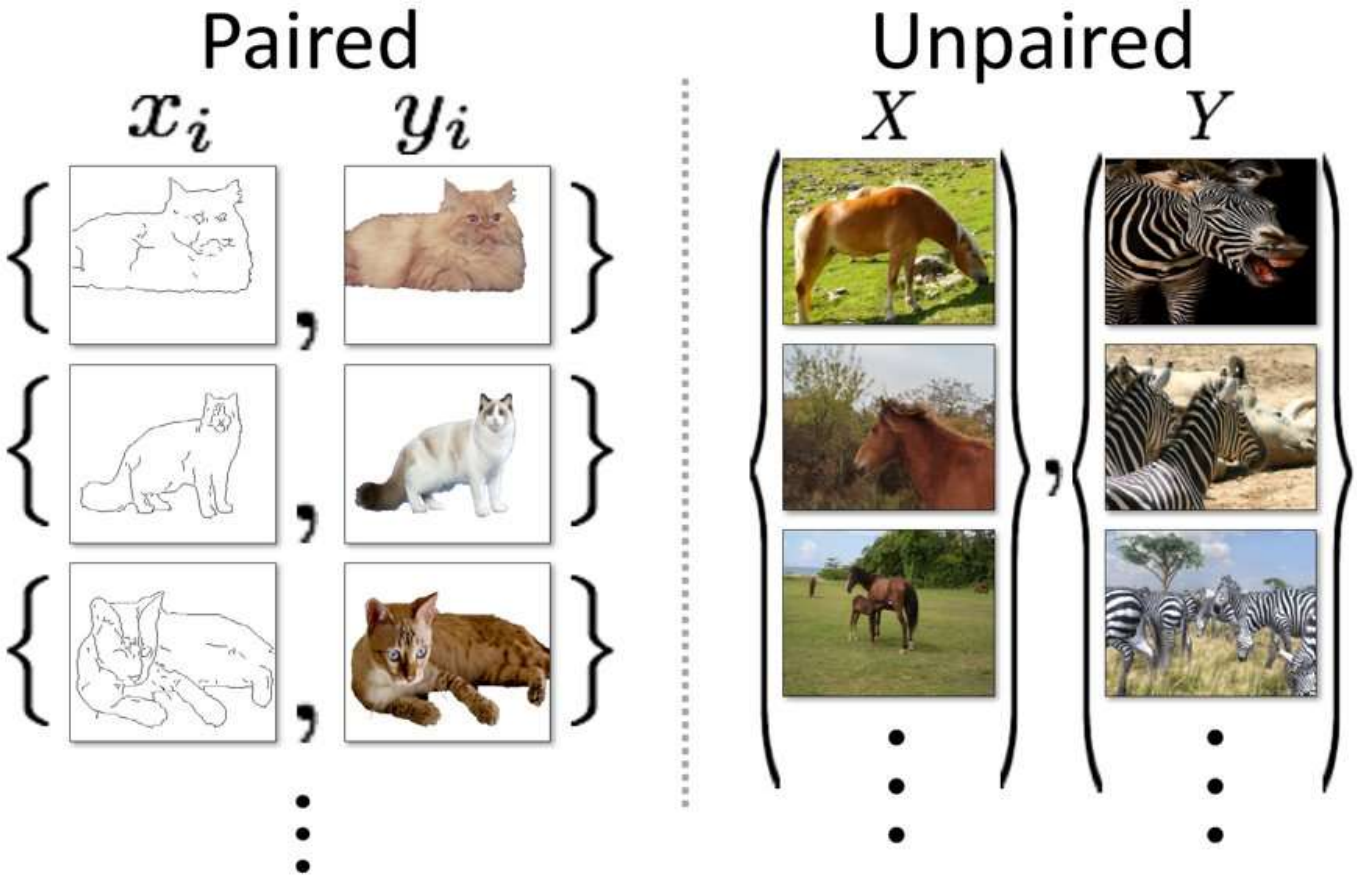
Label ↔ photo: per-pixel labeling



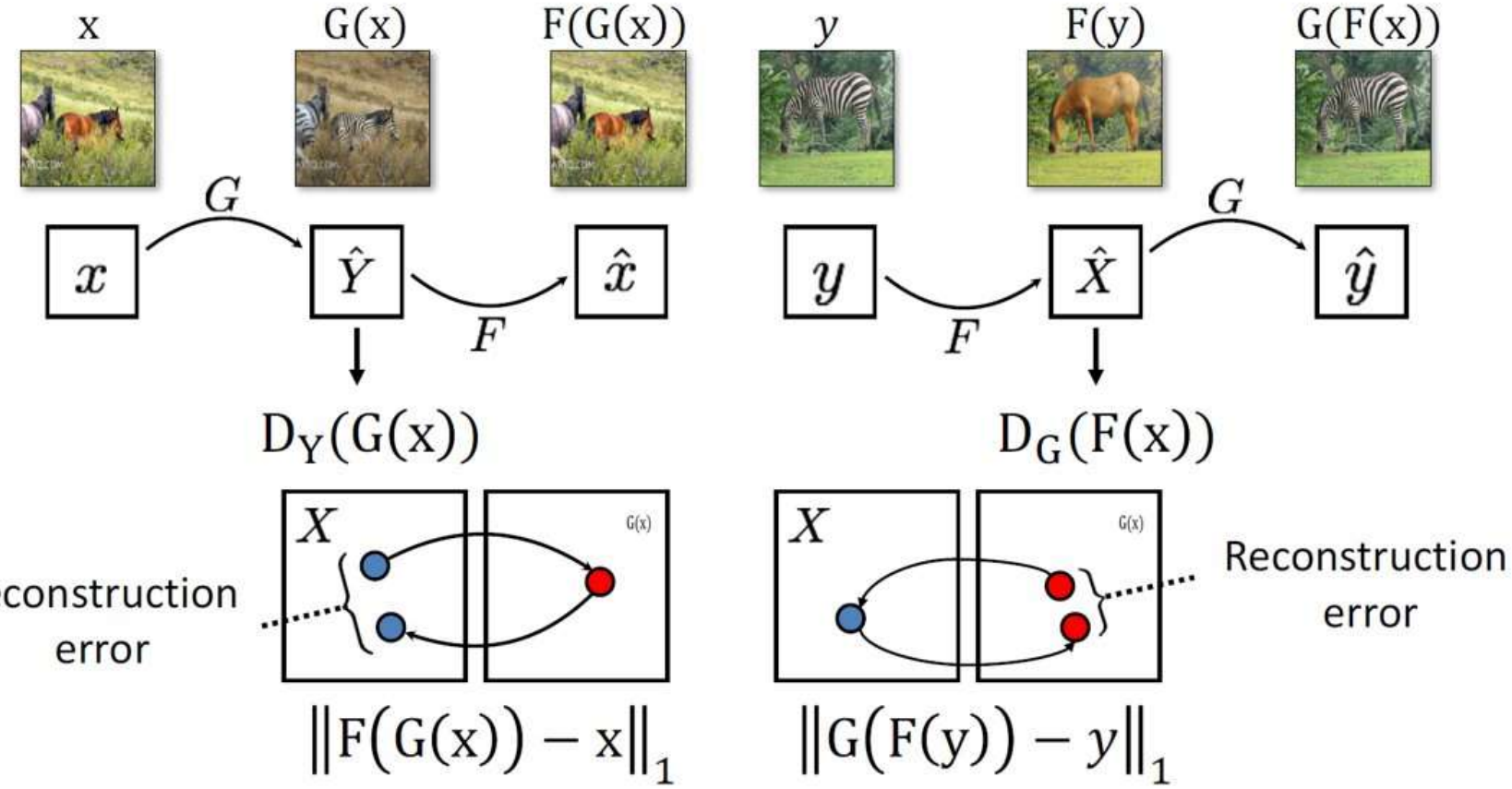
Horse ↔ zebra: how to get zebras?

- Expensive to collect pairs.
- Impossible in many scenarios.

UNPAIRED TRAINING – MORE PRACTICAL



CYCLE CONSISTENCY LOSS – USE 2 GANS



[IASI AI] CYCLEGAN - UNPAIRED IMAGE-TO-IMAGE TRANSLATION

While an *x* and *y* set of images are still required, they do not need to directly correspond to each other
It uses *two* generators and *two* discriminators. Both G and F are generators that take an image from one domain and translate it to another. One discriminator provides adversarial training for G, and the other does the same for F.

$$G : X \rightarrow Y \qquad D_y : \text{Distinguishes y from G(x)}$$
$$F : Y \rightarrow X \qquad D_x : \text{Distinguishes x from F(y)}$$

The Objective Function
The adversarial loss enforces that the generated output be of the appropriate domain but does *not* enforce that the input and output are recognizably the same. It uses least squares loss which is more effective than the typical log likelihood loss (paper):

$$Loss_{adv}(G, D_y, X) = \frac{1}{m}\sum_{i=1}^{m}(1 - D_y(G(x_i)))^2$$

Cycle consistency loss - enforces that *F* (*G* (*x*)) ≈ *x* and *G* (*F* (*y*)) ≈ *y:*

$$Loss_{cyc}(G, F, X, Y) = \frac{1}{m}\sum_{i=1}^{m}[F(G(x_i)) - x_i] + [G(F(y_i)) - y_i]$$

Full loss function (lambda = hyperparameter):

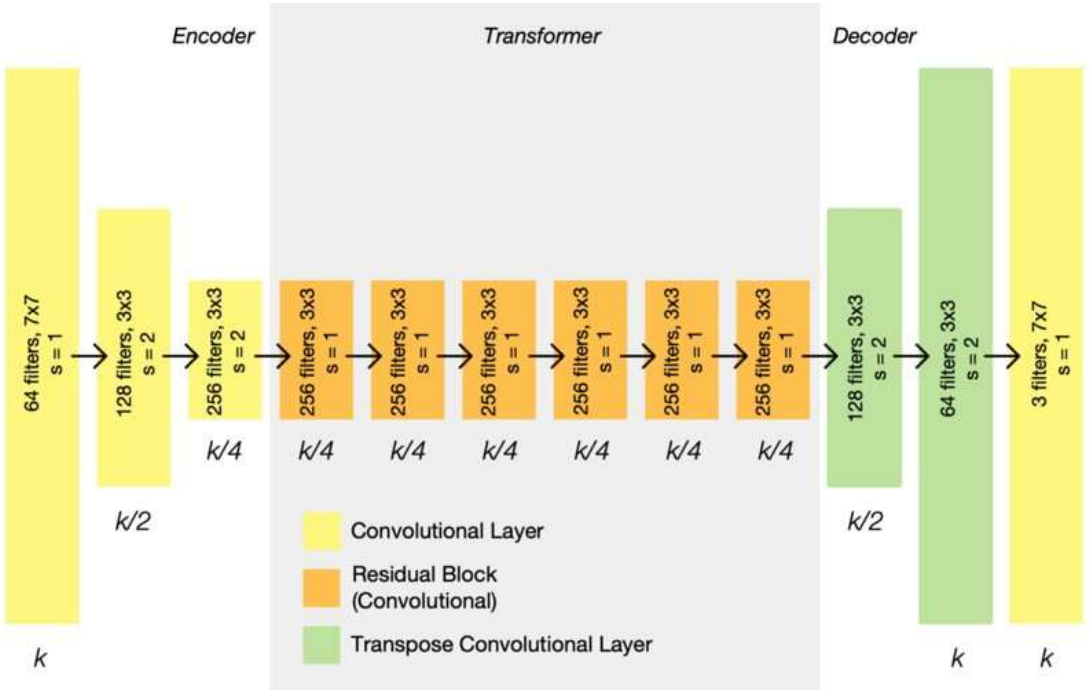
$$Loss_{full} = Loss_{adv} + \lambda Loss_{cyc}$$

CycleGAN: Learning to Translate Images (Without Paired Training Data)

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (2017 Berkeley), Code

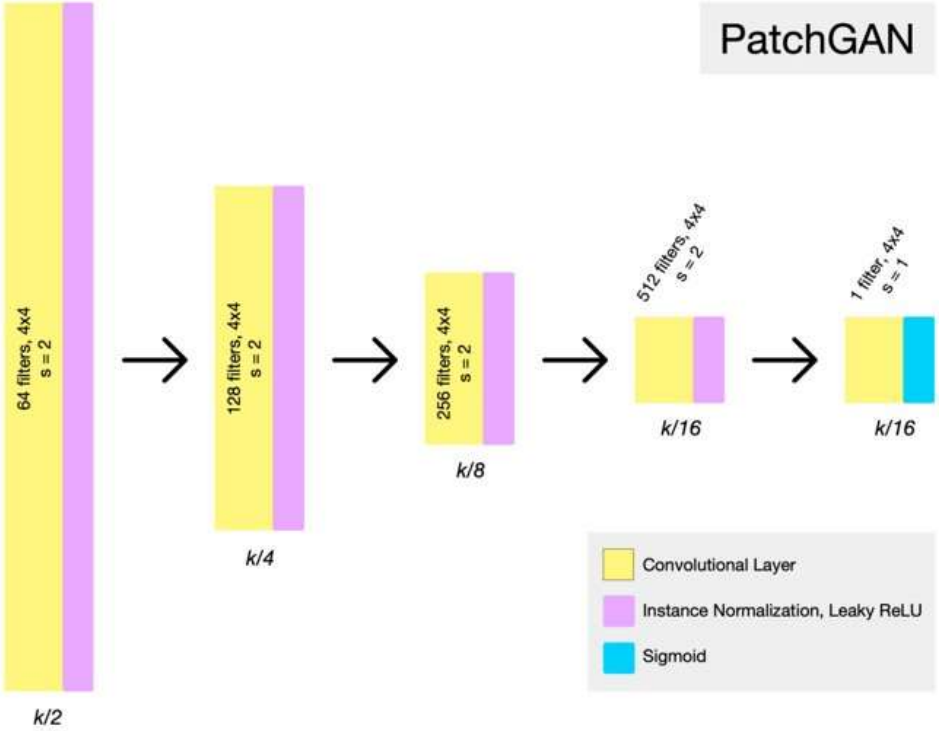
[IASI AI]

CYCLEGAN GENERATOR ARCHITECTURE



The representation size shrinks in the encoder phase, stays constant in the transformer phase, and expands again in the decoder phase. The representation size that each layer outputs is listed below it, in terms of the input image size, k. On each layer is listed the number of filters, the size of those filters, and the stride. Each layer is followed by an instance normalization and ReLU activation.

CYCLEGAN DISCRIMINATOR ARCHITECTURE



PatchGAN = fully convolutional neural network that look at a “patch” of the input image and output the probability of the patch being “real”. This is both more computationally efficient than trying to look at the entire input image and is also more effective

Structured prediction
It allows the discriminator to focus on more surface-level features, like texture, which is often the sort of thing being changed in an image translation task.

Implementing Spatial Batch / Instance / Layer Normalization in Tensorflow

It's a fully convolutional network, that takes in an image, and produces a matrix of probabilities, each referring to the probability of the corresponding “patch” of the image being “real” (as opposed to generated). The representation size that each layer outputs is listed below it, in terms of the input image size, k. On each layer is listed the number of filters, the size of those filters, and the stride.

[IASI AI]

RECYCLE-GAN: UNSUPERVISED VIDEO RETARGETING

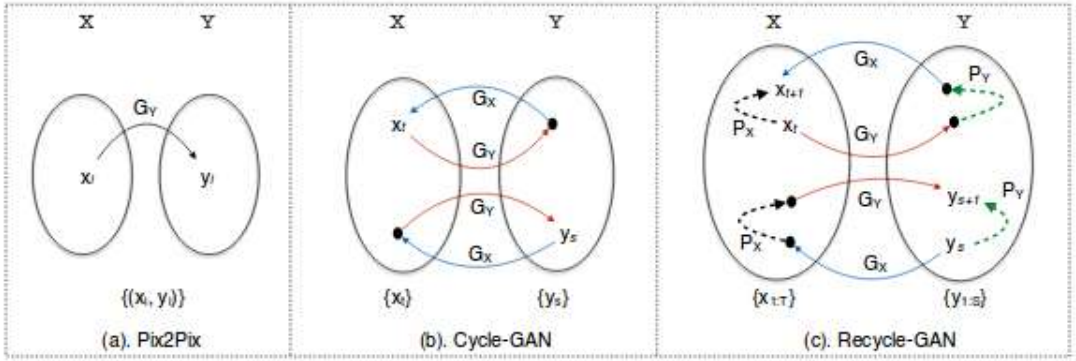


Fig. 3. We contrast our work with two prominent directions in image-to-image translation. (a) **Pix2Pix** [23]: Paired data is available. A simple function (Eq. 1) can be learnt via regression to map $X \rightarrow Y$. (b) **Cycle-GAN** [53]: The data is not paired in this setting. Zhu et al. [53] proposed to use cycle-consistency loss (Eq. 3) to deal with the problem of unpaired data. (c) **Recycle-GAN**: The approaches so far have considered independent 2D images only. Suppose we have access to unpaired but *ordered streams* $(x_1, x_2, \ldots, x_t, \ldots)$ and $(y_1, y_2 \ldots, y_s, \ldots)$. We present an approach that combines spatiotemporal constraints (Eq. 5). See Section 3 for more details.

Recurrent loss:

$$L_r(P_X) = \sum_t ||x_{t+1} - P_X(x_{1:t})||^2$$

Recycle loss (indirect):

$$L_r(G_X, G_Y, P_Y) = \sum_t ||x_{t+1} - G_X(P_Y(G_Y(x_{1:t})))||^2$$

Beyond Deep Fakes: Transforming Video Content Into Another Video's Style, Automatically

Recycle-GAN: Unsupervised Video Retargeting, Behavioral imitation using Recycle-GAN, Steve Buscemi + Jennifer Lawrence MASHUP

INSTAGAN - INSTANCE-AWARE IMAGE-TO-IMAGE TRANSLATION (2019)

Previous methods (CycleGAN) often fail in challenging cases, when an image has multiple target instances and a translation task involves significant changes in shape, e.g., translating pants to skirts in fashion images.
=> We propose InstaGAN, that incorporates the instance information (e.g., object segmentation masks) and improves multi-instance transfiguration. The proposed method translates both an image and the corresponding set of instance attributes while maintaining the permutation invariance property of the instances.

Github implementation + paper

[IASI AI] FROM GAN TO SAGAN

Model	Inception Score	FID
AC-GAN [31]	28.5	/
SNGAN-projection [17]	36.8	27.62*
SAGAN	**52.52**	**18.65**

- Depth and Convolution
- Class-conditional generation
- Spectral Normalization

They applied spectral normalization to the weights in both generator and discriminator. They set the spectral norm to 1 to constrain the Lipschitz constant of the weights. **It's just used for controlling the gradients**

- Hinge loss
- Two-timescale update rule

They used a two-timescale update rule (TTUR) which is simply using different learning rate for both discriminator and generator.

- Self-attention

They used this self-attention layer in both the generator and discriminator

Soves the problem that generated images of dogs have missing legs etc.

[IASI AI]

SAGAN – RESULTS

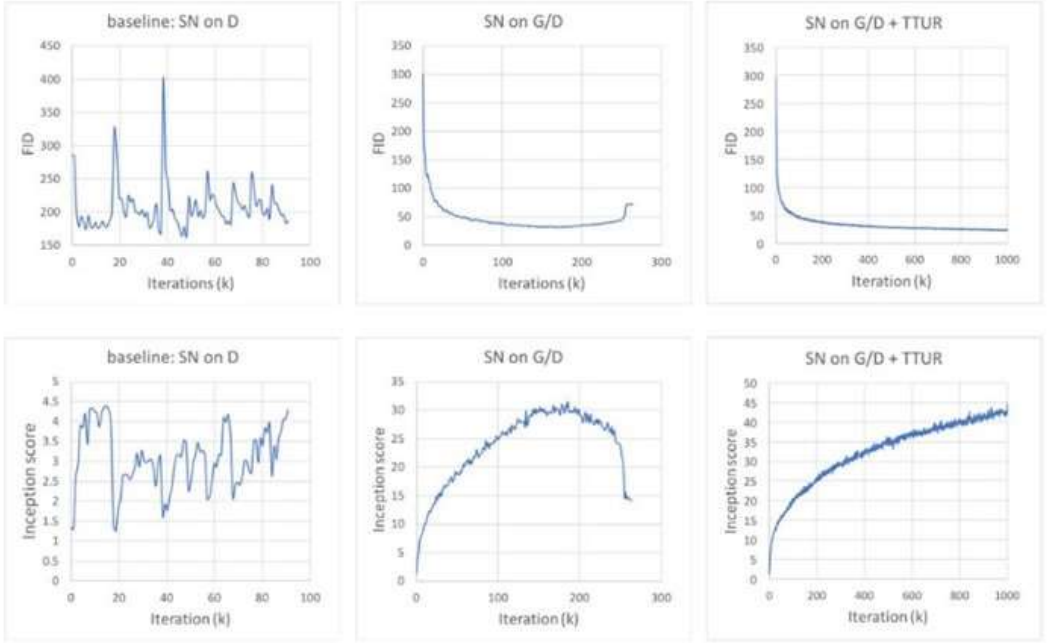


Figure 3: Training curves for the baseline model and our models with the proposed stabilization techniques, "SN on G/D" and two-timescale learning rates (TTUR). All models are trained with 1:1 balanced updates for G and D.

PROGRESSIVE GROWING OF GANS (PROGAN) - TOWARD HIGHER IMAGE RESOLUTION

NVIDIA presented the stunningly detailed 1024x1024 images generated by their new ProGAN:

Progressive growing of GANs
The unprecedented effectiveness of “Progressive Growing of GANs”
ProGAN: How NVIDIA Generated Images of Unprecedented Quality

PROGRESSIVE GROWING OF GANS (PROGAN)

Curriculum learning
ProGAN starts out generating very low-resolution images. When training stabilizes, a new layer is added, and the resolution is doubled. This continues until the output reaches the desired resolution. By progressively growing the networks in this fashion, high-level structure is learned first, and training is stabilized. ProGAN generally trained about 2–6 times faster than a corresponding traditional GAN, depending on the output resolution.

“Fading in” New Layers
To prevent shocks in the pre-existing lower layers from the sudden addition of a new top layer, the top layer is linearly “faded in”. This fading in is controlled by a parameter *α*, which is linearly interpolated from 0 to 1 over the course of many training iterations.

DCGAN used ***transpose convolutions*** to change the representation size. In contrast, ProGAN uses *nearest neighbors* for upscaling and ***average pooling*** for downscaling. These are simple operations with no learned parameters. They are then followed by two convolutional layers.

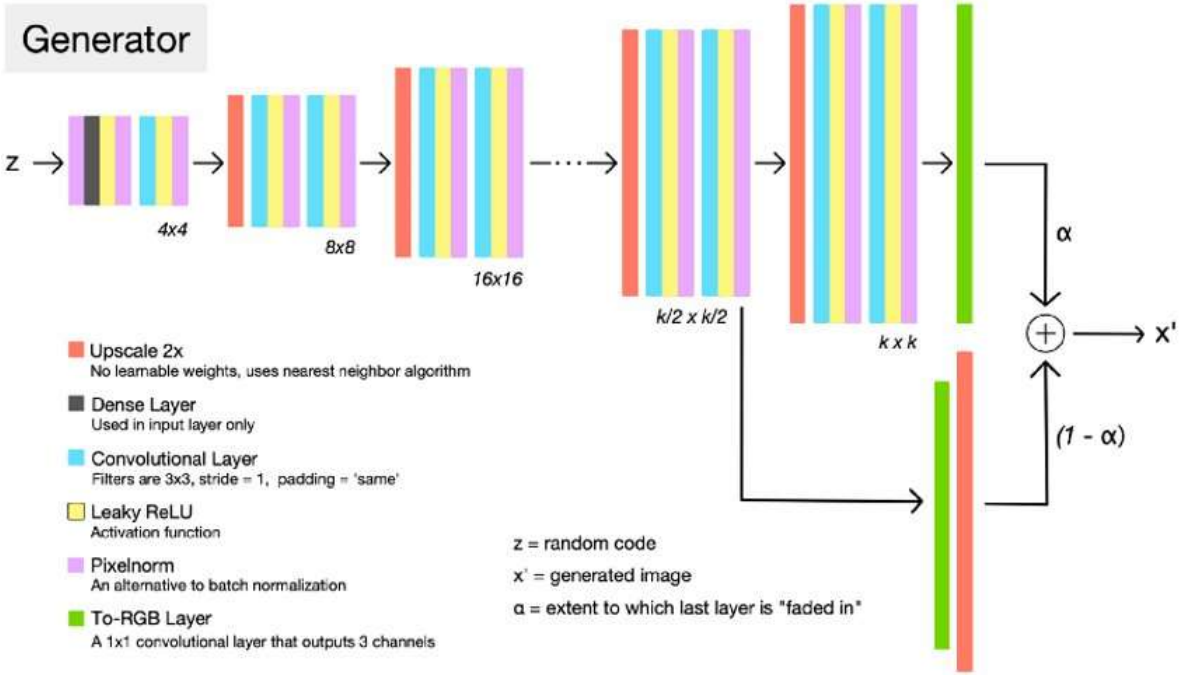
Pixel Normalization
Instead of using batch normalization, as is commonly done, the authors used *pixel normalization*.

$$b_{x,y} = \frac{a_{x,y}}{\sqrt{\frac{1}{C}\sum_{j=0}^{C} a_{x,y}^{j} + \epsilon}}$$

The values of each pixel (x, y) across C channels are normalized to a fixed length. Here, **a** is the input tensor, **b** is the output tensor, and **ε** is a small value to prevent dividing by zero.

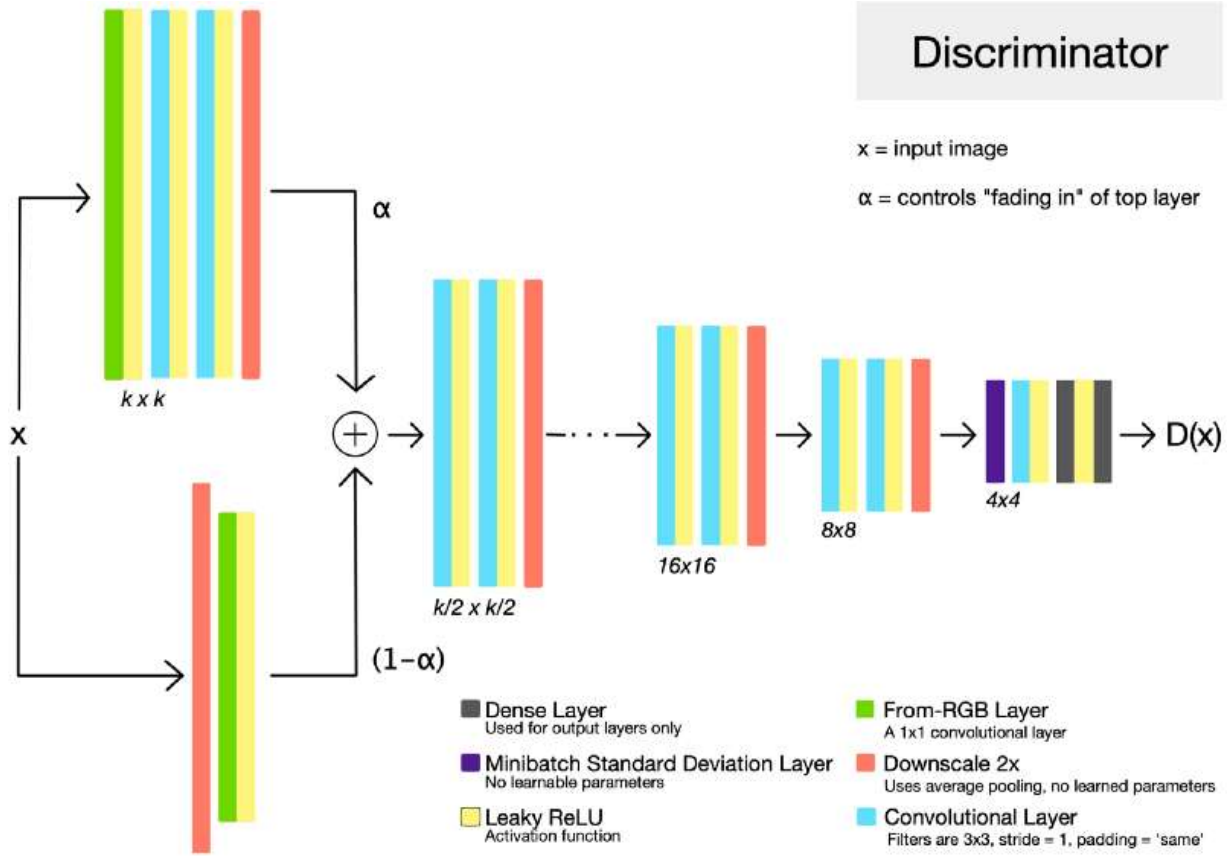
[IASI AI]

PROGAN – GENERATOR ARCHITECTURE



A detailed view of the generator architecture, when it has “grown” to resolution k. Each set of layers doubles the resolution size with a nearest neighbor upscaling operation followed by two convolutions. To stabilize training, the most recently added layer is “faded in”. This process is controlled by α, a number between 0 and 1 that is linearly increased over many training iterations until the new layer is fully in place.

PROGAN – DISCRIMINATOR ARCHITECTURE



A detailed view of the discriminator architecture, when it has “grown” to resolution k.

Here, x is the input image (either generated or from the training set), α is the extent to which the last generator layer is “faded in”, and D(x) is the probability the generator has assigned to x being from the training set.

The representation size is halved at each set of layers by an average pooling operation.

[IASI AI]

PROGAN – MORE TRICKS

Minibatch Standard Deviation

- encourages the generator to produce more variety, such that statistics computed across a generated batch more closely resemble those from a training data batch
- this is done by inserting a “minibatch standard deviation” layer near the end of the discriminator. This layer has no trainable parameters. It computes the standard deviations of the feature map pixels across the batch, and appends them as an extra channel

Equalized Learning Rate

- to ensure healthy competition between the G and D, it is essential that layers learn at a similar speed. To achieve this *equalized learning rate*, they scale the weights of a layer according to how many weights that layer has.
- For example, before performing a convolution with f filters of size [k, k, c], we would scale the weights of those filters as shown below:

$$W_f = W_i * \sqrt{\frac{2}{k * k * c}}$$

- **Use Improved Wasserstein loss function (**Wasserstein GAN with gradient penalty (WGAN-GP)):

$$Loss_G = -D(x')$$

$$GP = (\|\nabla D(ax' + (1-a)x))\|_2 - 1)^2$$

$$Loss_D = -D(x) + D(x') + \lambda * GP$$

[IASI AI]

BIGGAN - A NEW STATE OF THE ART IN IMAGE SYNTHESIS

BigGAN = 512×512 resolution class-conditional GAN trained from ImageNet dataset, generate image corresponding to the label based on input noise + image label

Improves Inception Score (IS) with 100% : 52.52 - >166.3. Frechet Inception Distance (FID) score = 18.65 -> 9.6

Model	Res.	FID/IS	(min FID) / IS	FID / (valid IS)	FID / (max IS)
SN-GAN	128	27.62 / 36.80	N/A	N/A	N/A
SA-GAN	128	18.65 / 52.52	N/A	N/A	N/A
BigGAN	128	$8.7 \pm .6/98.8 \pm 2.8$	$7.7 \pm .1/126.5 \pm .1$	$9.6 \pm .4/166.3 \pm 1$	$25 \pm 2/206 \pm 2$
BigGAN	256	$8.2 \pm .2/154 \pm 2.5$	$7.7 \pm .1/178 \pm 5$	$9.3 \pm .3/233 \pm 1$	$25 \pm 5/295 \pm 4$
BigGAN	512	10.9 / 154.9	9.3 / 202.5	10.9 / 241.4	24.4/275

Table 2: Evaluation of models at different resolutions. We report scores without truncation (Column 3), scores at the best FID (Column 4), scores at the IS of validation data (Column 5), and scores at the max IS (Column 6). Standard deviations are computed over at least three random initializations.

We show that GANs benefit dramatically from scaling, and train models with two to four times as many parameters and eight times the batch size compared to prior art.
We introduce two simple, general architectural changes that improve scalability, and modify a regularization scheme to improve conditioning, demonstrably boosting performance.
As a side effect of our modifications, our models become amenable to the “truncation trick,” a simple sampling technique that allows explicit, fine-grained control of the tradeoff between sample variety and fidelity

Large Scale GAN Training for High Fidelity Natural Image Synthesis, BigGAN: A New State of the Art in Image Synthesis , Colab

BIG GAN- SCALING UP GANS

They simply increase the batch size and the number of channels in the network layer. The results are shown below. The blue frame is 8 times the batch size from the baseline (256 → 2048), and the green frame is further 1.5 times the number of channels from all layers. Finally, with this only change, you can see that the score is updated significantly.

In the paper, they improve the score further by adding three more methods. That is **Shared Embedding**, **Hierarchical latent spaces**, **Orthogonal Regularization** (corresponding to Shared, Hier., Ortho. In the table below)

Batch	Ch.	Param (M)	Shared	Hier.	Ortho.	Itr $\times 10^3$	FID	IS
256	64	81.5	SA-GAN Baseline			1000	18.65	52.52
512	64	81.5	✗	✗	✗	1000	15.30	58.77(±1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03(±1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85(±3.83)
2048	96	173.5	✗	✗	✗	295(±18)	9.54(±0.62)	92.98(±4.27)
2048	96	160.6	✓	✗	✗	185(±11)	9.18(±0.13)	94.94(±1.32)
2048	96	158.3	✓	✓	✗	152(±7)	8.73(±0.45)	98.76(±2.84)
2048	96	158.3	✓	✓	✓	165(±13)	8.51(±0.32)	99.31(±2.10)
2048	64	71.3	✓	✓	✓	371(±7)	10.48(±0.10)	86.90(±0.61)

Table 1: Fréchet Inception Distance (FID, lower is better) and Inception Score (IS, higher is better) for ablations of our proposed modifications. *Batch* is batch size, *Param* is total number of parameters, *Ch.* is the channel multiplier representing the number of units in each layer, *Shared* is using shared embeddings, *Hier.* is using a hierarchical latent space, *Ortho.* is Orthogonal Regularization, and *Itr* either indicates that the setting is stable to 10^6 iterations, or that it collapses at the given iteration. Other than rows 1-4, results are computed across 8 different random initializations.

BIGGAN - HIERARCHICAL LATENT SPACES WITH SHARED EMBEDDINGS

In a typical cGAN, input noise is given only to the first layer, but here it first splits the input noise and gives it to each ResBlock.

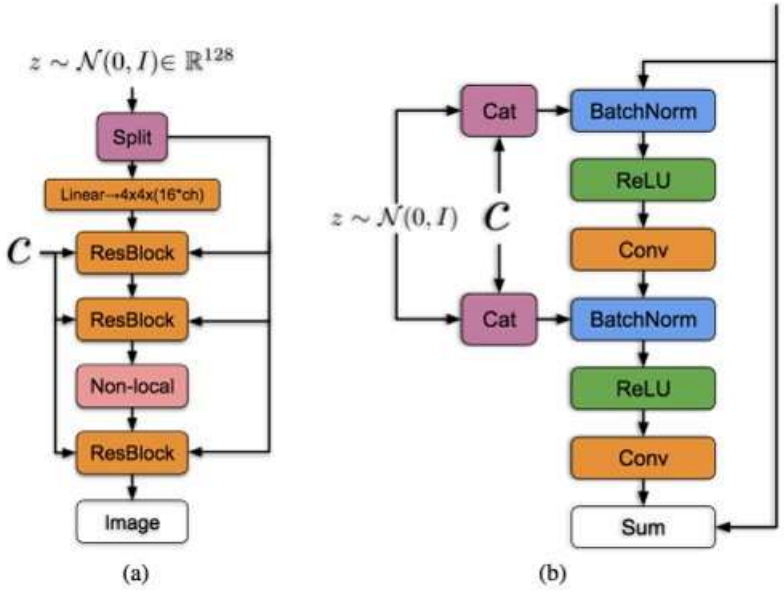


Figure 15: (a) A typical architectural layout for **G**; details are in the following tables. (b) A Residual Block in **G**. c is concatenated with a chunk of z and projected to the BatchNorm gains and biases.

The intuition behind this design is to allow G to use the latent space to directly influence features at different resolutions and levels of hierarchy. For our architecture, this is easily accomplished by splitting z into one chunk per resolution, and concatenating each chunk to the conditional vector c which gets projected to the BatchNorm gains and biases.

We note that class embeddings c used for the conditional BatchNorm layers in G contain a large number of weights. Instead of having a separate layer for each embedding (Miyato et al., 2018; Zhang et al., 2018), we opt to use a shared embedding, which is linearly projected to each layer's gains and biases (Perez et al., 2018).

BIGGAN - TRADING OFF VARIETY AND FIDELITY WITH THE NOISE TRUNCATION TRICK

- Taking a model trained with $z \sim N(0, I)$ and sampling z from a truncated normal (where values which fall outside a range are resampled to fall inside that range) immediately provides a boost to IS and FID.

- We call this the Truncation Trick: truncating a z vector by resampling the values with magnitude above a chosen threshold leads to improvement in individual sample quality at the cost of reduction in overall sample variety.

(a) (b)

Figure 2: (a) The effects of increasing truncation. From left to right, threshold=2, 1, 0.5, 0.04. (b) Saturation artifacts from applying truncation to a poorly conditioned model.

[IASI AI]

BIGGAN - ORTHOGONAL REGULARIZATION TO MAKE THE MODEL SUITABLE FOR TRUNCATION

- Orthogonality is a desirable quality in ConvNet filters, partially because multiplication by an orthogonal matrix leaves the norm of the original matrix unchanged. This property is valuable in deep or recurrent networks, where repeated matrix multiplication can result in signals vanishing or exploding. (Original Paper) (Reddit)
- We augment our objective with the cost $\mathcal{L}_{ortho} = \Sigma(|WW^T - I|)$

where Σ indicates a sum across all filter banks, W is a filter bank, and I is the identity matrix.

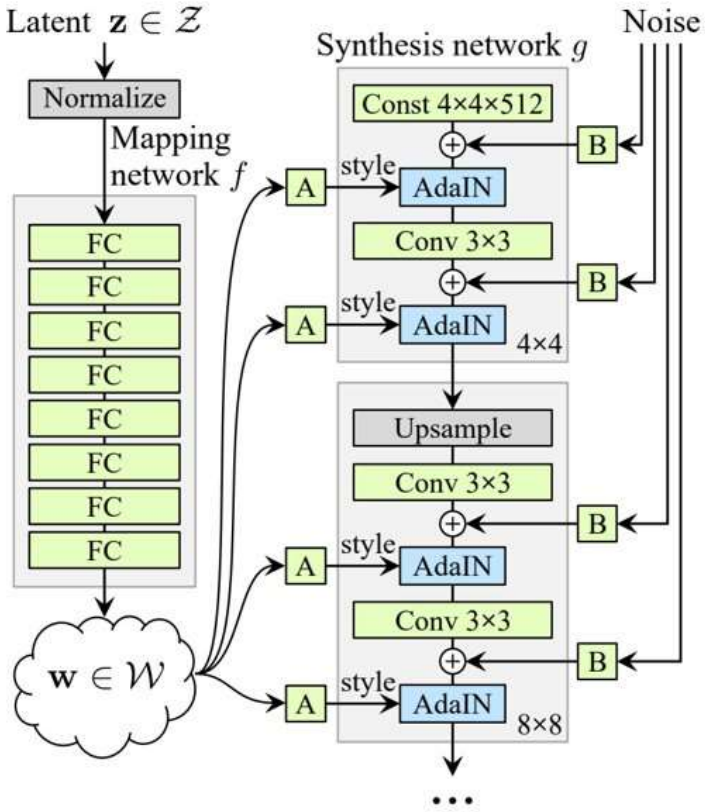
Some math: By multiplying with orthogonal matrices we preserve length of a vector or matrix
(MIT Orthogonal Matrices Course)

To fix saturation artifacts we seek to enforce amenability to truncation by conditioning G to be smooth, so that the full space of z will map to good output samples. They use a modified version of Orthogonal Regularization since the original one is too limiting:

$$R_\beta(W) = \beta\|W^\top W - I\|_{\mathrm{F}}^2 \quad \Rightarrow \quad R_\beta(W) = \beta\|W^\top W \odot (\mathbf{1} - I)\|_{\mathrm{F}}^2,$$

It removes the diagonal terms from the regularization, and aims to minimize the pairwise cosine similarity between filters but does not constrain their norm

STYLE GAN - A STYLE-BASED GENERATOR ARCHITECTURE



A very novel solution to the disentanglement problem:
- learn abstractions about hair, eyes, smiles, skin color, race, pose as styles
- learn the codes to these styles

Generator learns a new encoding (**w**) through using an 8 layer fully connected (MLP) network. This encoding is used as input to learn different styles (**A**) at different levels of resolution. The AdaIN blocks are style transfer networks. So the difference here between the Progressive GAN network is in the inclusion of a coding component and a style component.

Here "A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input

To quantify interpolation quality and disentanglement, the paper proposes two new, automated methods—**perceptual path length and linear separability**—that are applicable to any generator architecture.

StyleGAN, Why Stylistic GANs are So Deceptive, GAN 2.0: NVIDIA's Hyperrealistic Face Generator

[IASI AI] WASSERSTEIN GAN

It is one of the most popular GANs and consists of an objective change which results in training stability, interpretability (correlation of the losses with sample quality) and the ability of generating categorical data.

For training loss estimation, It uses **Earth-Mover (EM) distance / Wasserstein Metric** (instead of KL divergence) to measure the distance between the 2 distributions (real and fake data) .

Advantages:

Correlation between loss metric and image quality

- In GAN, the G loss measures how well it fools the discriminator rather than a measure of the image quality. The G loss does not drop even the image quality improves. Hence, we cannot tell the progress from its value. We need to save the testing images and evaluate it visually.
- On the contrary, WGAN loss function reflects the image quality which is more desirable.

Improve training stability

- it has no sign of mode collapse in experiments
- the generator can still learn when the critic perform well.

Good for categorical data: provides the GAN with the ability of generating *categorical* data (i.e., not continuous-valued data like images or even integer-coded data like 1 for Sunday, 2 for Monday and so on) (*"penguin + .001" ? Fake for sure)*

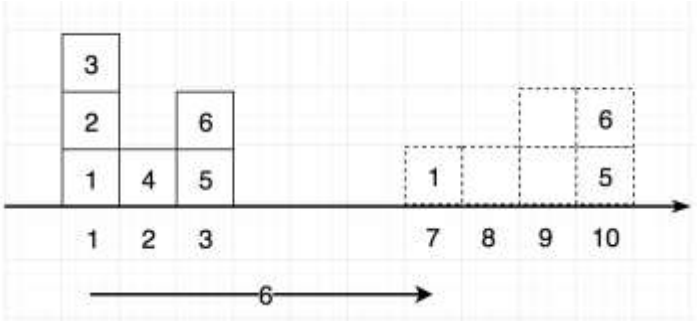
Note: Usually G Cannot be differentiable if output is discrete.

Why Wasserstein is better than JS or KL divergence (Math), WGAN & WGAN-GP, Towards data set augmentation with GANs

[IASI AI]

EARTH-MOVER (EM) DISTANCE / WASSERSTEIN METRIC

We get 6 boxes and we want to move them from the left to the locations marked by the dotted square on the right. The moving cost equals to its weight times the distance. For simplicity, we will set the weight to be 1.



The cost to move box #1 equals to 6 (7–1)

Example when probability domain is *discrete:*
P and Q, each has four piles of dirt and both have ten shovelfuls of dirt in total. The numbers of shovelfuls in each dirt pile are assigned as follows:

$$P_1 = 3, P_2 = 2, P_3 = 1, P_4 = 4$$
$$Q_1 = 1, Q_2 = 2, Q_3 = 4, Q_4 = 3$$

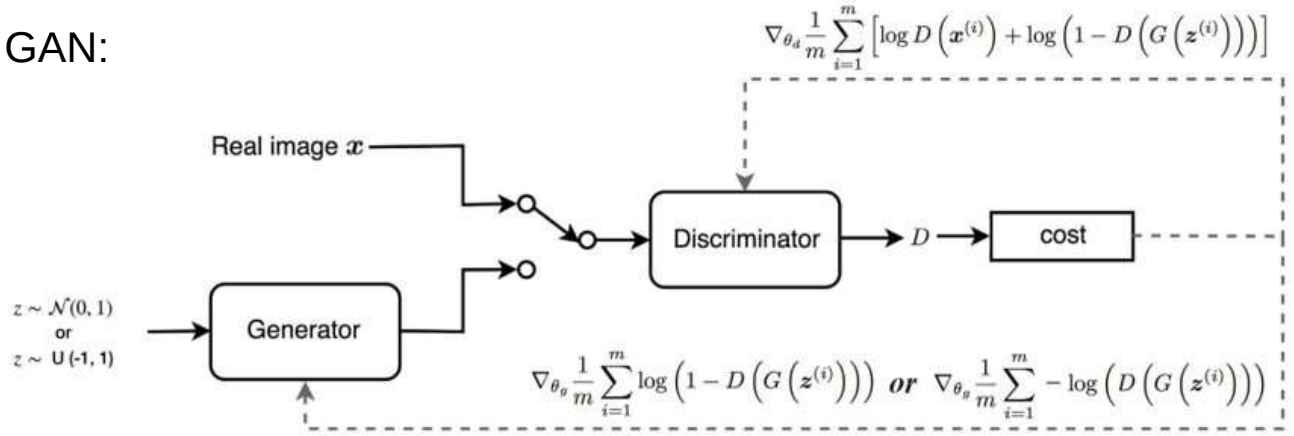
(….) => Wasserstein distance can be approximated by training a critic. D learns a K-Lipschitz continuous function to do this:

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

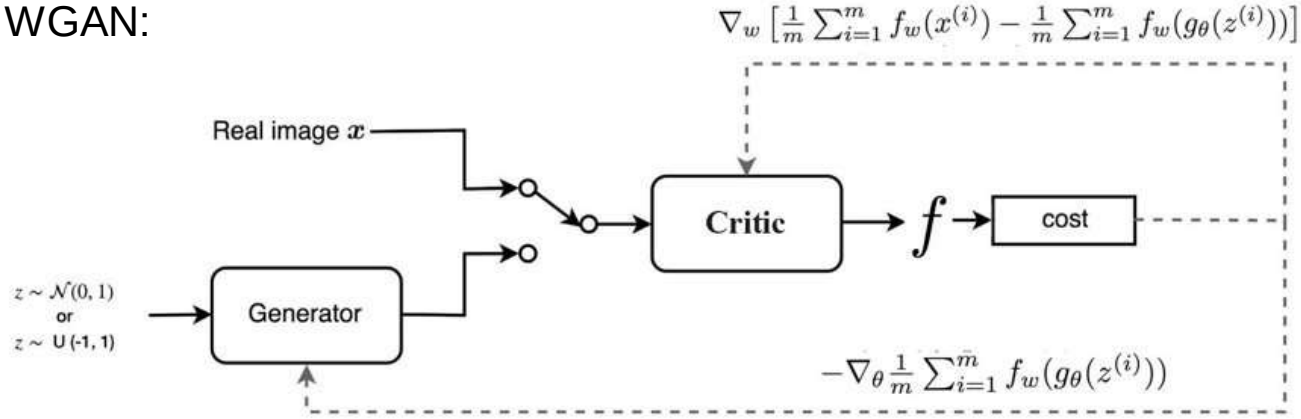
Why Wasserstein is better than JS or KL divergence (Math), WGAN & WGAN-GP, Towards data set augmentation with GANs

[IASI AI] GAN VS WGAN ARHITECTURE

GAN:



WGAN:



f is a 1-Lipschitz function:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

The critic learns f.
It is very similar to ***D,*** just without the sigmoid function and outputs a scalar score rather than a probability. This score can be interpreted as how real the input images are.

To enforce 1-Lipschitz condition WGAN applies a very simple clipping to restrict the maximum weight value in ***f***, i.e. the weights of the discriminator must be within a certain range controlled by the hyperparameters ***c***

$$w \leftarrow w + \alpha \cdot \mathrm{RMSProp}(w, g_w)$$
$$w \leftarrow \mathrm{clip}(w, -c, c)$$

WGAN TRAINING ALGORITHM

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

[IASI AI]

WGAN ISSUES – CLIPPING IS TOO LIMITING

The difficulty in WGAN is to enforce the Lipschitz constraint.

Clipping is simple but it introduces some problems:

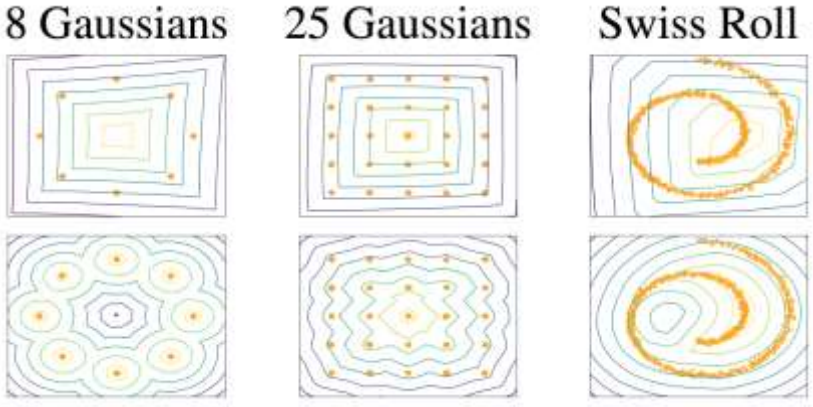
"Weight clipping is a clearly terrible way to enforce a Lipschitz constraint" (Oops!). WGAN still suffers from unstable training,

Slow convergence after weight clipping (when clipping window is too large) and vanishing gradients (when clipping window is too small).

The model performance is very sensitive to this clipping hyperparameter c.

Model capacity too limited

The weight clipping behaves as a weight regulation. It reduces the capacity of the model ***f*** and limits the capability to model complex functions.



The reduced capacity of WGAN fails to create a complex boundary to surround the modes (orange dots) of the model while the improved WGAN-GP can (second row).

WGAN-GP (WGAN WITH GRADIENT PENALTY)

It uses another way to enforce 1-Lipschtiz.

The major advantage of WGAN-GP is its convergence. It makes training more stable and therefore easier to train. As WGAN-GP helps models to converge better, we can use a more complex model like a deep ResNet for the generator and the discriminator

For WGAN, the gradient is smoother everywhere and learns better even the generator is not producing good images..

A differentiable function ***f*** is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere.

So instead of applying clipping, WGAN-GP penalizes the model if the gradient norm moves away from its target norm value 1

f^* has gradient norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$.

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\left[D(\tilde{\boldsymbol{x}})\right] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\left[D(\boldsymbol{x})\right]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[\left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)^2\right]}_{\text{Our gradient penalty}}.$$

where $\hat{\boldsymbol{x}}$ sampled from $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}$ with t uniformly sampled between 0 and 1

$$\hat{\boldsymbol{x}} = t\tilde{\boldsymbol{x}} + (1-t)\boldsymbol{x} \text{ with } 0 \le t \le 1$$

[IASI AI]

0 CENTERED GRADIENT PENALTY IS BETTER (2019)

- D trained with the original GAN loss have poor generalization capability. Poor generalization in the discriminator prevents the generator from learning the target distribution.
- Original GAN objective encourages gradient exploding in the discriminator. Gradient exploding in the discriminator can lead to mode collapse in the generator.
- Experiments on synthetic and real-world datasets verify that 0-GP can prevent mode collapse. GANs with 0-GP are much more robust to changes in hyper parameters, optimizers, and network architectures than the original GAN and GANs with other gradient penalties.

GP	Formula	Improve generalization	Prevent grad expoding	Convergence guarantee
Our 0-GP	$\lambda \mathbb{E}_{\boldsymbol{v} \in \mathcal{C}}[\|(\nabla D)_{\boldsymbol{v}}\|^2]$, $\mathcal{C}$ from $\boldsymbol{y}$ to $\boldsymbol{x}$	✓	✓	✓
1-GP	$\lambda \mathbb{E}_{\tilde{\boldsymbol{x}}}[(\|(\nabla D)_{\tilde{\boldsymbol{x}}}\| - 1)^2]$, where $\tilde{\boldsymbol{x}} = \alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y}$	✗	✓	✗
0-GP-sample	$\lambda \mathbb{E}_{\boldsymbol{v} \in \mathcal{D}}[\|(\nabla D)_{\boldsymbol{v}}\|^2]$	✗	✗	✓

$$\mathcal{L}_{0-GP} = \mathcal{L} - \lambda \mathbb{E}_{\tilde{\boldsymbol{x}}}\left[\left\|(\nabla D)_{\tilde{\boldsymbol{x}}}\right\|^2\right] \text{ where } \tilde{\boldsymbol{x}} = \alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y},\ \boldsymbol{x} \sim p_r,\ \boldsymbol{y} \sim p_g, \text{ and } \alpha \sim \mathcal{U}(0,1).$$

The remaining problem is how to find the path C from a fake to a real sample which lies inside supp(pr) ∪ supp(pg).
We approximate C with the straight line connecting a pair of samples.

Improving Generalization and Stability of Generative Adversarial Networks

WGAN-GP TRAINING ALG

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

[IASI AI]

SPECTRAL NORM REGULARIZATION – BETTER NETWORK GENERALIZATION

Better model generalization = a model that is insensitive to the small perturbations of the input = wide minima.

Our goal is to obtain a model, Θ for model f, such that the f(x + ξ) − f(x) L2-norm is small,
where x is an arbitrary input vector and ξ is a perturbation vector with a small L2-norm.

if we consider a small neighborhood of x, we can regard f as a linear function (we use RELU activation)
In other words, we can represent it by an affine map, x → W*x + b

$$\frac{\|f_\Theta(\boldsymbol{x}+\boldsymbol{\xi})-f(\boldsymbol{x})\|_2}{\|\boldsymbol{\xi}\|_2} = \frac{\|(W_{\Theta,\boldsymbol{x}}(\boldsymbol{x}+\boldsymbol{\xi})+\boldsymbol{b}_{\Theta,\boldsymbol{x}})-(W_{\Theta,\boldsymbol{x}}\boldsymbol{x}+\boldsymbol{b}_{\Theta,\boldsymbol{x}})\|_2}{\|\boldsymbol{\xi}\|_2} = \frac{\|W_{\Theta,\boldsymbol{x}}\boldsymbol{\xi}\|_2}{\|\boldsymbol{\xi}\|_2} \leq \sigma(W_{\Theta,\boldsymbol{x}})$$

The *spectral norm* of a matrix $A \in \mathbb{R}^{m\times n}$ is defined

$$\sigma(A) = \max_{\boldsymbol{\xi}\in\mathbb{R}^n,\boldsymbol{\xi}\neq\mathbf{0}} \frac{\|A\boldsymbol{\xi}\|_2}{\|\boldsymbol{\xi}\|_2}$$

We refer to the second term as the spectral norm regularizer. It decreases the spectral norms of the weight matrices.

$$\underset{\Theta}{\text{minimize}}\ \frac{1}{K}\sum_{i=1}^{K} L(f_\Theta(\boldsymbol{x}_i),\boldsymbol{y}_i) + \frac{\lambda}{2}\sum_{\ell=1}^{L}\sigma(W^\ell)^2$$

Spectral Norm Regularization for Improving the Generalizability of Deep Learning

[IASI AI]

SPECTRALLY NORMALIZED GAN - SN-GAN

- Novel weight normalization method to stabilize the training of discriminator networks
- Better than previous regularization methods (weight normalization , weight clipping, gradient penalty, orthonormal regularization)
- More stable GAN training
- Simple algorithm & computational efficient
- Lipschitz constant is the only hyper-parameter to be tuned
- Capable of generating images of better or equal quality
- Spectral normalization is normalizing the weights of a discriminator with their spectral norm so that the Lipschitz norm of the discriminator to be bounded by 1

"The Lipschitz constant of a linear function is its largest singular value, or its spectral norm" => normalize weight matrix by its spectral norm

Fixing the spectral norm of a layer is as straightforward as it sounds. *Spectral normalization*, proposed in this paper, simply replaces every weight W with $W/\sigma(W)$. But how do we efficiently compute $\sigma(W)$, the largest singular value of W? The answer is a cheap and effective technique called *power iteration*.

Spectral Normalization Explained

Spectral Normalization for Generative Adversarial Networks , (Paper) ,(Code)

SGD WITH SPECTRAL NORMALIZATION

Algorithm 1 SGD with spectral normalization

- Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
- For each update and each layer l:
  1. Apply power iteration method to a unnormalized weight W^l:

$$\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l\|_2 \tag{20}$$

$$\tilde{\boldsymbol{u}}_l \leftarrow W^l \tilde{\boldsymbol{v}}_l / \|W^l \tilde{\boldsymbol{v}}_l\|_2 \tag{21}$$

  2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:

$$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}} W^l \tilde{\boldsymbol{v}}_l \tag{22}$$

  3. Update W^l with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate α:

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M) \tag{23}$$

[IASI AI] VECTOR NORM FORMULAS

In 2-dimensional space is vector length:

Definition: A norm $\|\cdot\|$ is a function from $\mathbb{C}^n$ into $\mathbb{R}$ such that:

1. $\|x\| \geq 0, \|x\| = 0 \Leftrightarrow x = 0$ — Lengh = 0 => Zero vector
2. $\|x + y\| \leq \|x\| + \|y\|$ — *triangle inequality*
3. $\|\alpha x\| = |\alpha| \cdot \|x\|$ — *absolutely scalable*

Pitagora formula: $\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}$

Norm = root ((x transposed) * x) : $\|\boldsymbol{x}\| := \sqrt{\boldsymbol{x}^* \, \boldsymbol{x}}$

Norm = Inner product of x with itself: $\|\boldsymbol{x}\| := \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$

Dot product: $\vec{a} \cdot \vec{b} = a_x \cdot b_x + a_y \cdot b_y = |\vec{a}||\vec{b}| \cos(\theta)$

[IASI AI]

MATRIX EIGENVECTORS & EIGENVALUES

An **eigenvector** or **characteristic vector** of a linear transformation is a non-zero vector that changes by only a scalar factor (eigenvalue) when that linear transformation is applied to it:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Geometrically, an eigenvector, corresponding to a real nonzero eigenvalue, points in a direction that is stretched by the transformation and the eigenvalue is the factor by which it is stretched. (Wiki)

If you can draw a line through the three points $(0,0)$, v and Av, then Av is just v multiplied by a number λ; that is, $Av = \lambda v$. In this case, we call λ an **eigenvalue** and v an **eigenvector**. For example, here $(1,2)$ is an eigvector and 5 an eigenvalue.

$$Av = \begin{pmatrix} 1 & 2 \\ 8 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 5 \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \lambda v.$$

Eigenvectors and Eigenvalues Explained Visually

SINGULAR VALUE DECOMPOSITION (SVD)

Now comes a highlight of linear algebra. Any real $m \times n$ matrix can be factored as

$$A = U\Sigma V^T$$

where U is an $m \times m$ orthogonal matrix[1] whose columns are the eigenvectors of AA^T, V is an $n \times n$ orthogonal matrix whose columns are the eigenvectors of A^TA, and Σ is an $m \times n$ diagonal matrix of the form

$$\Sigma = \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & 0 & \\ & & \sigma_r & & \\ & & & 0 & \\ & 0 & & & \ddots \\ & & & & & 0 \end{pmatrix}$$

with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and $r = \text{rank}(A)$. In the above, $\sigma_1, \ldots, \sigma_r$ are the square roots of the eigenvalues of A^TA. They are called the *singular values* of A.

[IASI AI]

POWER ITERATION ALGORITHM

Since SVD is slow we can use Power iteration algorithm to find only maximum singular value.

If A is an $n \times n$ diagonalizable matrix with a dominant eigenvalue, then there exists a nonzero vector $\mathbf{x}_0$ such that the sequence of vectors given by

$$A\mathbf{x}_0, \quad A^2\mathbf{x}_0, \quad A^3\mathbf{x}_0, \quad A^4\mathbf{x}_0, \quad \ldots, \quad A^k\mathbf{x}_0, \quad \ldots$$

approaches a multiple of the dominant eigenvector of A.

By the spectral theorem, we can write W in an orthonormal basis of eigenvectors of T(W) * W …

We iteratively perform the following procedure enough times:

$$\tilde{\boldsymbol{v}} \leftarrow W^{\mathrm{T}}\tilde{\boldsymbol{u}}/\|W^{\mathrm{T}}\tilde{\boldsymbol{u}}\|_2, \ \tilde{\boldsymbol{u}} \leftarrow W\tilde{\boldsymbol{v}}/\|W\tilde{\boldsymbol{v}}\|_2.$$

$$\sigma(W) = \sqrt{\lambda_1} = ||Wv||. \qquad \sigma(W) = ||Wv|| = u^T W v. \qquad \sigma(W) \approx \tilde{\boldsymbol{u}}^{\mathrm{T}} W \tilde{\boldsymbol{v}}.$$

Power method for aproximating eigenvalues
Spectral Normalization Explained , Paper

WHICH TRAINING METHODS FOR GANS DO CONVERGE?

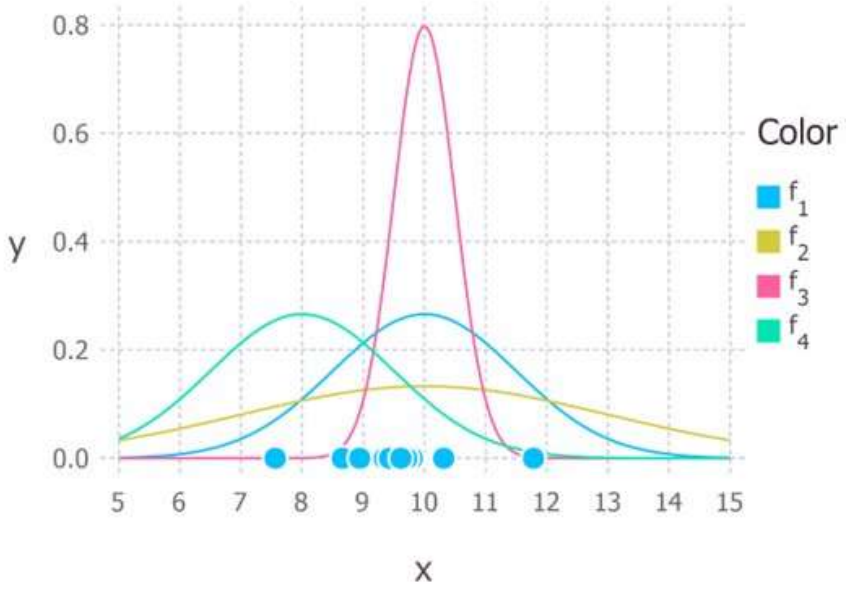
Method	Local convergence (a.c. case)	Local convergence (general case)
unregularized (Goodfellow et al., 2014)	✓	✗
WGAN (Arjovsky et al., 2017)	✗	✗
WGAN-GP (Gulrajani et al., 2017)	✗	✗
DRAGAN (Kodali et al., 2017)	✓	✗
Instance noise (Sønderby et al., 2016)	✓	✓
ConOpt (Mescheder et al., 2017)	✓	✓
Gradient penalties (Roth et al., 2017)	✓	✓
Gradient penalty on real data only	✓	✓
Gradient penalty on fake data only	✓	✓

Table 1. Convergence properties of different GAN training algorithms for general GAN-architectures. Here, we distinguish between the case where both the data and generator distributions are absolutely continuous (a.c.) and the general case where they may lie on lower dimensional manifolds.

[IASI AI]

MAXIMUM LIKELIHOOD ESTIMATION (OF DATA)

Here blue gaussian curve is best estimate of seen data.



$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

In maximum likelihood estimation we want to maximize the total probability of the data.
We find optimal parameters for the model, so the model describes best the log probability density function for the concentration of examples in the training data.

We can also think of maximum likelihood estimation as minimizing the KL divergence between the data generating distribution and the model

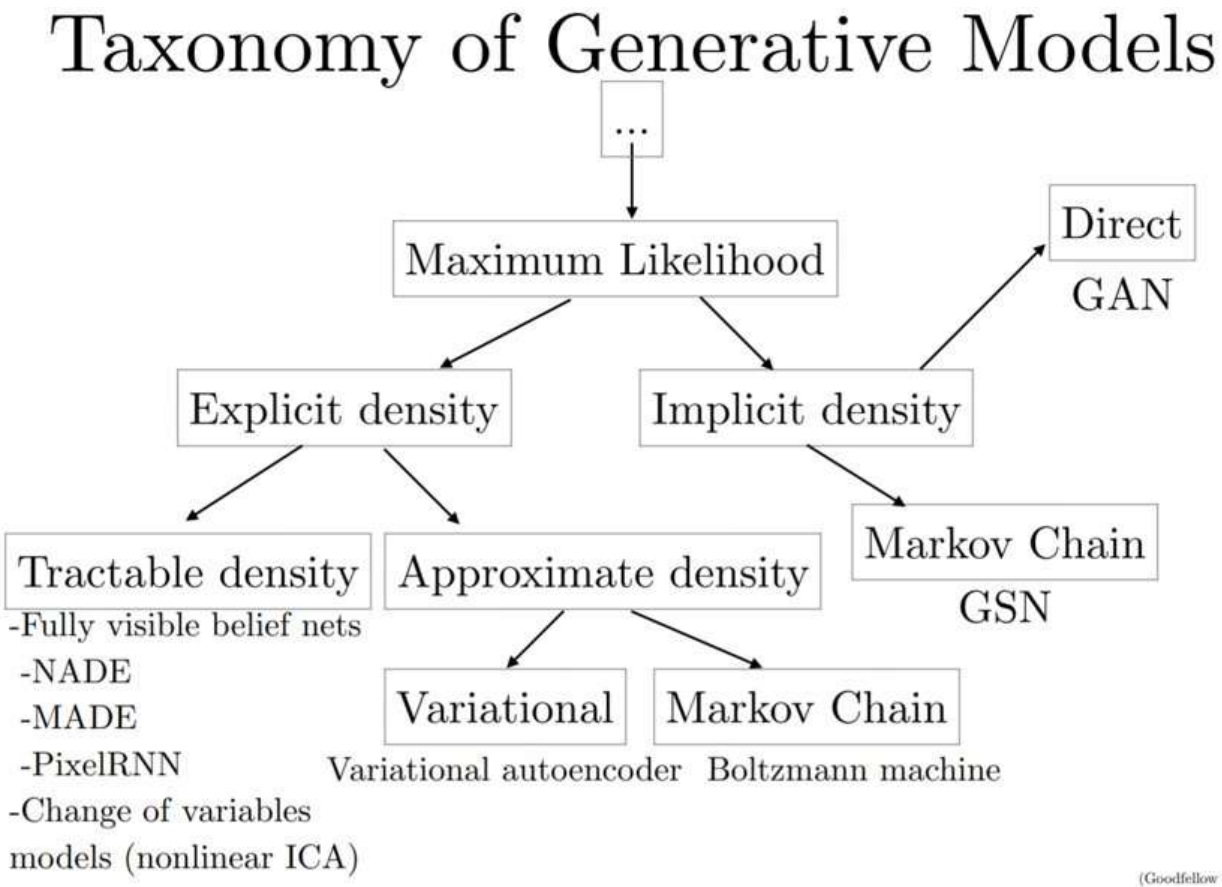
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} D_{\text{KL}} \left(p_{\text{data}}(\boldsymbol{x}) \| p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \right)$$

Note: The model could generate more data from a mix of examples with good probability.

Fun Math: Probability of a single point to happen is 0.
=> density function means probability of small region around the point

[IASI AI]

OLDER GENERATIVE MODELS BEFORE GAN



Various disadvantages:

- slow generation of samples
- lower quality
- do not scale for large dataset
- no latent code to use for embeddings
- cannot recover true distribution of data

Some have advantages over GAN:

VAEs have other problems, but do not suffer from mode-collapse
=>
DeepMind:
Combining VAEs and GANs :
VAE-GAN hybrid
What The Heck Are VAE-GANs?

[IASI AI]

KULLBACK–LEIBLER AND JENSEN–SHANNON DIVERGENCE

$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q\|\frac{p+q}{2})$$

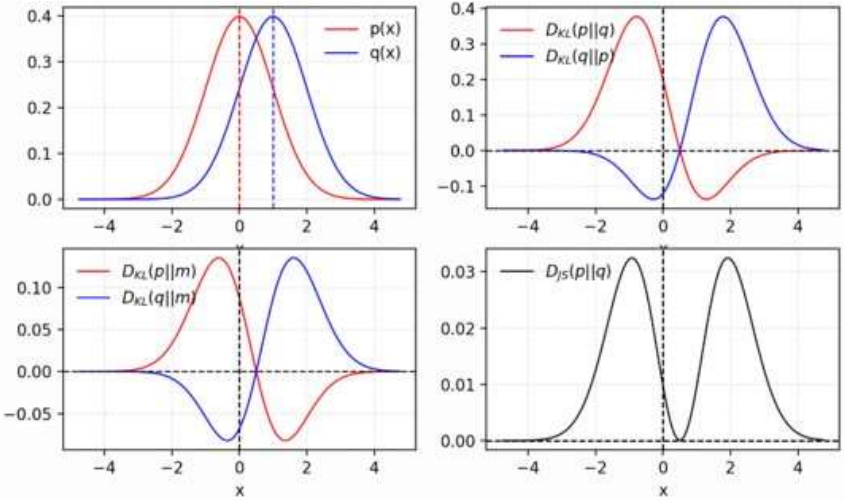


Fig. 1. Given two Gaussian distribution, p with mean=0 and std=1 and q with mean=1 and std=1. The average of two distributions is labelled as $m = (p+q)/2$. KL divergence D_{KL} is asymmetric but JS divergence D_{JS} is symmetric.

Original GAN uses JS divergence

From GAN to WGAN

[IASI AI] CONVERGENCE – DEEP LEARNING VS GANS

$$\min_G L(G)$$

Deep Learning models (in general) involve a single player - Convergence

- The player tries to maximize its reward (minimize its loss).
- Use SGD (with Backpropagation) to find the optimal parameters.
- SGD has convergence guarantees (under certain conditions).
- Problem: With non-convexity, we might converge to local optima.
- Optimization algorithms often approach a saddle point or local minimum rather than a global minimum

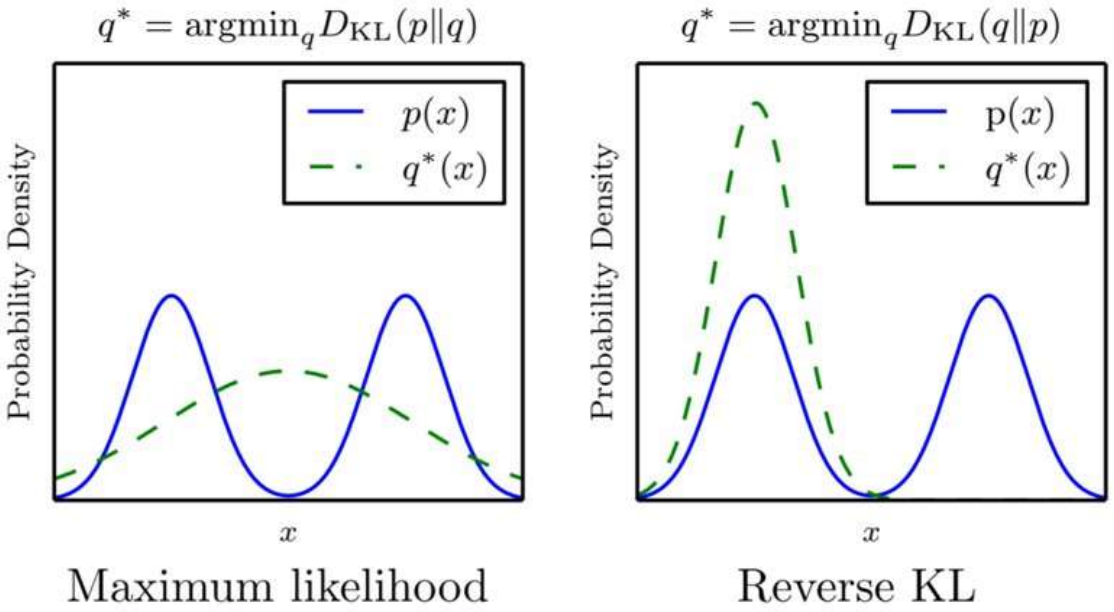
$$\min_G \max_D V(D,G)$$

GANs instead involve two players – Convergence not guaranteed

- Discriminator is trying to maximize its reward.
- Generator is trying to minimize Discriminator's reward.
- SGD was not designed to find the Nash equilibrium of a game.
- Problem: We might not converge to the Nash equilibrium at all.
- Game solving algorithms may not approach an equilibrium at all

[IASI AI]

IS THE DIVERGENCE IMPORTANT?



We try to learn a gaussian function to minimize difference of entropy vs true distribution. On the left learns both modes. On the right D learns only one mode but avoids the valley of untruth between the 2 modes.

Note: The original GAN formula resembles Jensen-Shannon divergence

[IASI AI]

LOSS DOES NOT SEEM TO EXPLAIN WHY GAN SAMPLES ARE SHARP

KL

Reverse KL

(Nowozin et al 2016)

KL samples from LSUN

REVERSE KL LOSS DOES NOT EXPLAIN MODE COLLAPSE

- Other GAN losses also yield mode collapse
- Reverse KL loss prefers to fit as many modes as the model can represent and no more; it does not prefer fewer modes in general
- GANs often seem to collapse to far fewer modes than the model can represent

Density Ratios and Classification

Density Ratio $\frac{p^*(\mathbf{x})}{q(\mathbf{x})}$

Bayes' Rule $p(\mathbf{x}|y) = \frac{p(y|\mathbf{x})p(\mathbf{x})}{p(y)}$

Combine data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} = \{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_{\hat{n}}, \tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{\tilde{n}}\}$

Assign labels $\{y_1, \ldots, y_N\} = \{+1, \ldots, +1, -1, \ldots, -1\}$

Real Data | Simulated Data

Computing a density ratio is equivalent to class probability estimation:

Density ratio

$$\frac{p^*(\mathbf{x})}{q(\mathbf{x})} = \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=-1)}$$

Bayes' substitution

$$= \frac{p(y=+1|\mathbf{x})p(\mathbf{x})}{p(y=+1)} \Big/ \frac{p(y=-1|\mathbf{x})p(\mathbf{x})}{p(y=-1)}$$

Class probability

$$\frac{p^*(\mathbf{x})}{q(\mathbf{x})} = \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})}$$

Understanding GANs

[IASI AI]

Thank You!

iasi.ai

meetup.com/IASI-AI/

fb.me/AI.Iasi/

We ❤ feedback!