Machine Intelligence & Deep Learning Workshop

Raymond Ptucha, Majid Rabbani, Mark Smith

The Kate Gleason COLLEGE OF ENGINEERING

Generative Adversarial Networks

Raymond Ptucha

June 27-29, 2018

R·I·T

Rochester Institute of Technology

www.rit.edu/kgcoe/cqas/machinelearning

Machine Intelligence Lab (MIL) Rochester Institute of Technology

Fair Use Agreement

This agreement covers the use of all slides in this document, please read carefully.

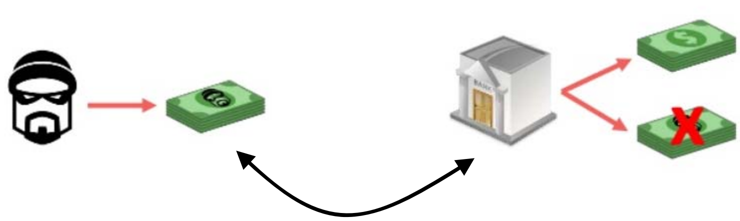
- You may freely use these slides for personal use, if:
  - My name (R. Ptucha) appears on each slide.
- You may freely use these slides externally, if:
  - You send me an email telling me the conference/venue/company name in advance, and which slides you wish to use.
  - You receive a positive confirmation email back from me.
  - My name (R. Ptucha) appears on each slide you use.

Agenda

- Wed, June 27
  - 9-10:30am — Regression and Classification
  - 10:30-10:45pm — Break
  - 10:45-12:15pm — Boosting and SVM
  - 12:15-1:30pm — Lunch
  - 1:30-3:30pm — Neural Networks and Dimensionality Reduction
  - 3:30-5pm — Hands-on Python and Machine Learning
- Thur, June 28
  - 9-10:30am — Introduction to deep learning
  - 10:30-10:45pm — Break
  - 10:45-12:15pm — Convolutional Neural Networks
  - 12:15-1:30pm — Lunch
  - 1:30-3:30pm — Region and pixel-level convolutions
  - 3:30-5pm — Hands-on CNNs
- Fri, June 29
  - 9-10:30am — Recurrent neural networks
  - 10:30-10:45pm — Break
  - 10:45-12:15pm — Language and Vision
  - 12:15-1:30pm — Lunch
  - 1:30-3:30pm — Graph convolutional neural networks; **Generative adversarial networks**
  - 3:30-5pm — Hands-on regional CNNs, RNNs

Intuition

- Bad guy analyzes real money and tries to make counterfeit bills
- Bank considers itself an expert at classifying money as real or counterfeit



Generator
Jointly learn prob of input and labels simultaneously P(x,y)

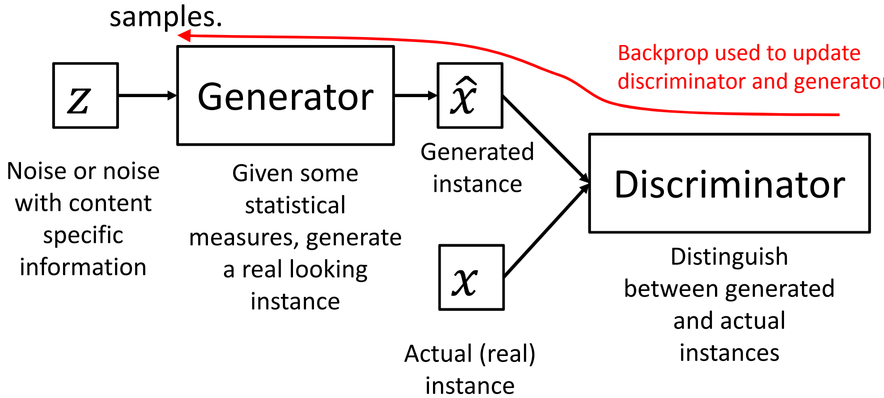
Two-player game

Discriminator
Given x, what is its class, P(y|x)

Images from https://www.slideshare.net/ThomasDaSilvaPaula/a-very-gentle-introduction-to-generative-adversarial-networks-aka-gans-71614428

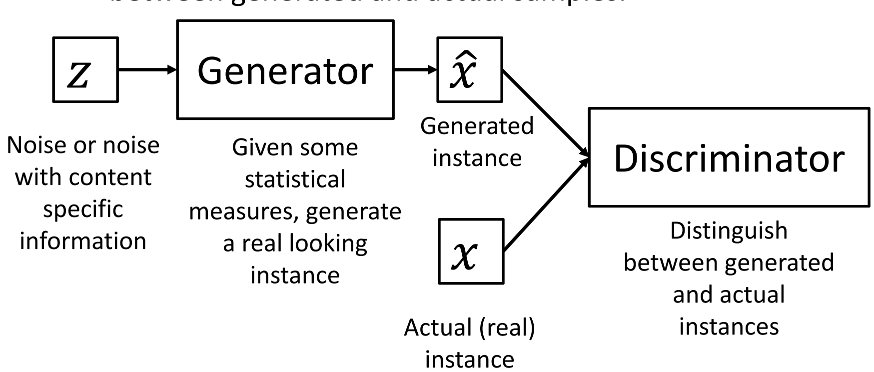
Structure of GANs

- Two competing neural networks:
  - Generator tries to generate realistic samples to fool discriminator.
  - Discriminator tries to distinguish between real and actual samples.



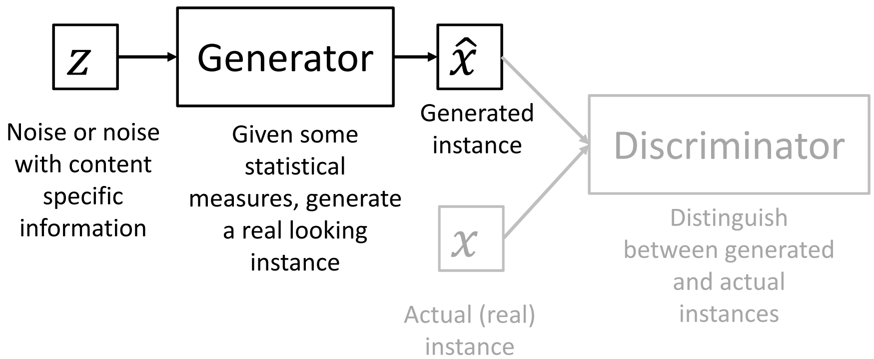
Structure of GANs

- We desire to continue training until:
  - The Generator exactly matches the true data representation.
  - The Discriminator can no longer tell the difference between generated and actual samples.



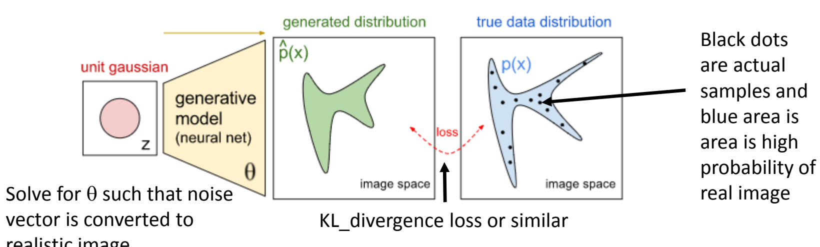
Structure of GANs

- When done, we generally toss the Discriminator.
- The Generator is now useful for data augmentation, unsupervised training, sample generation, sample understanding, …



What is Going On??

- Pick any domain, say images, sentences, sounds, etc; use millions of such samples to train a model to generate data to look as if it came from the original distribution.
- Unsupervised as no need for GT collection.
- Eventually GANs might be able to automatically discover and learn features of our world in an unsupervised fashion- once properly grouped only a few labelled exemplars are needed.

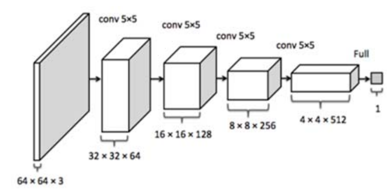


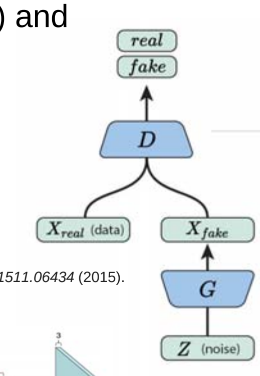
https://blog.openai.com/generative-models/

Generative Adversarial Networks (GANs) for Image Generation

- Jointly learn Discriminator (real vs. fake) and (image) Generator.

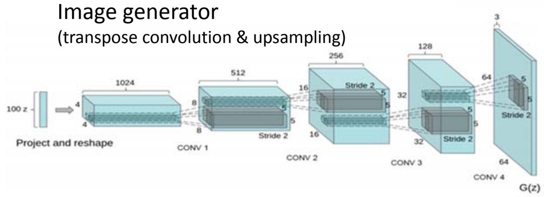
Discriminator





Radford, *arXiv:1511.06434* (2015).

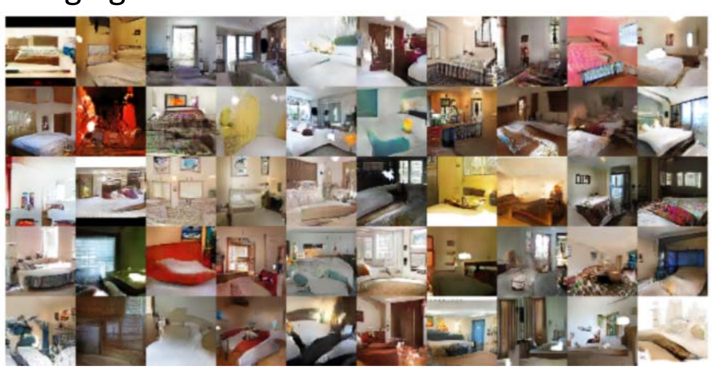
Image generator
(transpose convolution & upsampling)



Can generate image starting with noise

GANs For Image Generation

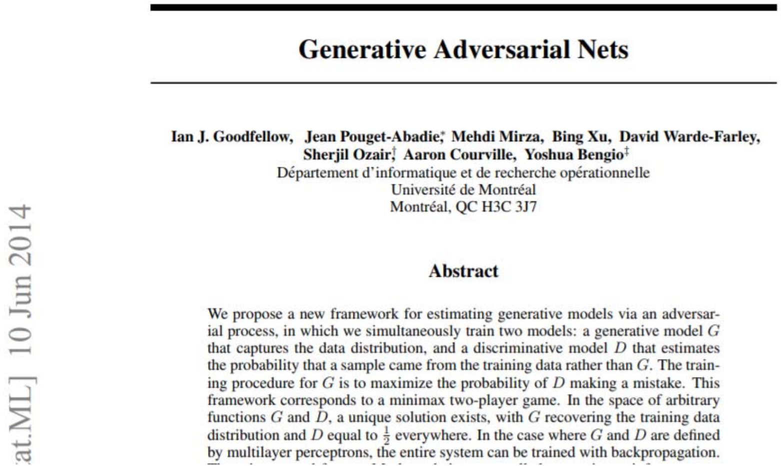
- GANs have outperformed other statistical methods at image generation.



"Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"

Seminal Paper

http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf



Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie*, Mehdi Mirza, Bing Xu, David Warde-Farley,
Sherjil Ozair†, Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

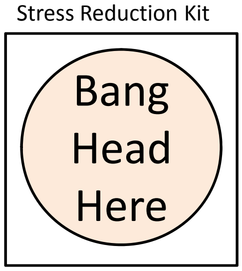
We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation.

Goodfellow et al. '14

- Train discriminator, *D* to maximize probability of detecting real vs. fake images.
- Train generator, *G* to minimize *log(1-D(G(z)))*.

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

Stress Reduction Kit



Kit Directions:
1. Place kit on firm surface
2. Follow directions in circle of kit
3. Repeat step 2 as necessary, or unconscious
4. If unconscious, cease stress reduction activity.

Goodfellow et al. '14

- Train discriminator, *D* to maximize probability of detecting real vs. fake images.
- Train generator, *G* to minimize *log(1-D(G(z)))*.

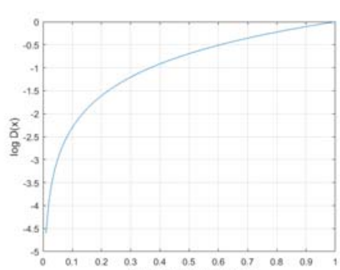
$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)]}_{\text{Real data}} + \underbrace{\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]}_{\text{Fake data}}$$

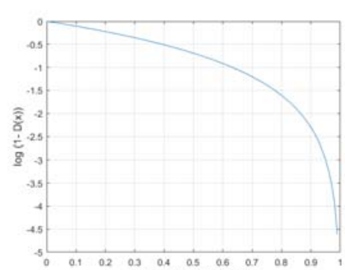
- Discriminator, *D* trained to output a 0 when fake input, 1 when real. Discriminator *D* output values in range {0:1}.
- Generator, *G* wants to trick discriminator, so *G* is trained such that when output of generator, *G(z)* is passed into discriminator *D(G(z))*, then the discriminator gets fooled and outputs a 1.

Goodfellow et al. '14

- Train discriminator, *D* to maximize probability of detecting real vs. fake images.
- Train generator, *G* to minimize *log(1-D(G(z)))*.

$$\min_G \max_D V(D,G) = \overbrace{\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)]}^{\text{Real data}} + \overbrace{\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]}^{\text{Fake data}}$$



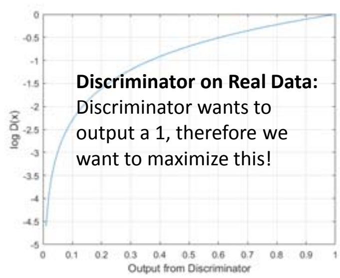


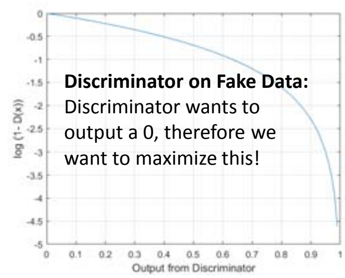
Goodfellow et al. ‘14

- Train discriminator, *D* to maximize probability of detecting real vs. fake images.
- Train generator, *G* to minimize *log(1-D(G(z)))*.

Real data Fake data

$$\min_{G} \max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$



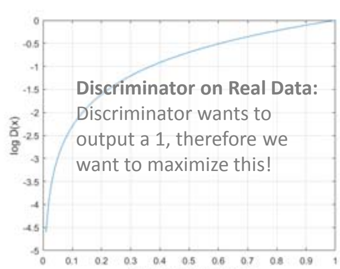


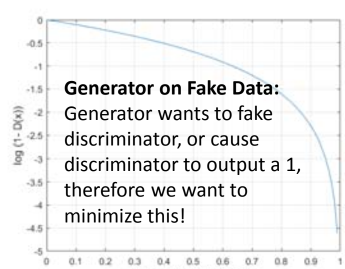
Goodfellow et al. ‘14

- Train discriminator, *D* to maximize probability of detecting real vs. fake images.
- Train generator, *G* to minimize *log(1-D(G(z)))*.

Real data Fake data

$$\min_{G} \max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$





Generator Loss in Practice

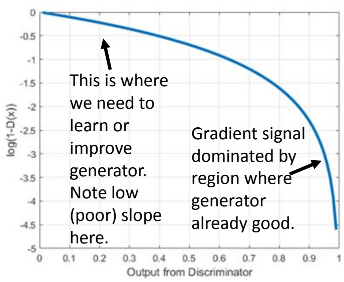
Note: this is for generator loss only- discriminator loss does not change.

- Instead of training generator, G to minimize *log(1-D(G(z)))*;
- train generator, G to maximize *-log(D(G(z)))*;

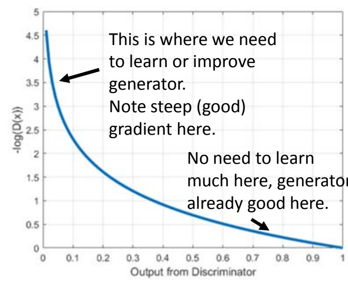
Real data　　　　Fake data

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

$$\max_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[-\log(D(G(z)))]$$



Generator on Fake Data:
Generator wants to fake discriminator, or cause discriminator to output a 1.



17

Training

for number of training iterations **do**

// Step 1: First train discriminator

for k steps **do**

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

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

Some folks use k=1, others use k>1. See Wasserstein GAN paper to get around this problem.

end for

// Step 2: Train generator

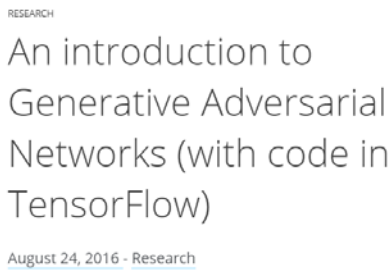
- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Stanford cs231n: Lecture 13

18



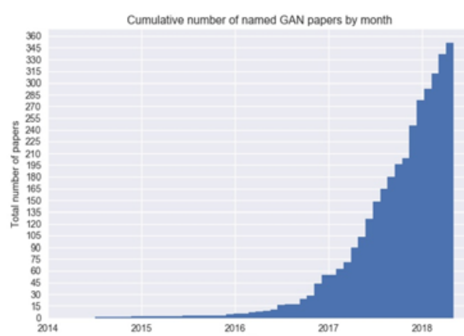
- Introductory description as well as TensorFlow code with 1D Gaussian example.
- Handful of existing GAN projects.

http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/

https://blog.openai.com/generative-models/

Other Good Resources

- Listing of hundreds of GAN papers:
  - The GAN Zoo- https://github.com/hindupuravinash/the-gan-zoo
- Tips and tricks for training GANs:
  - https://github.com/soumith/ganhacks



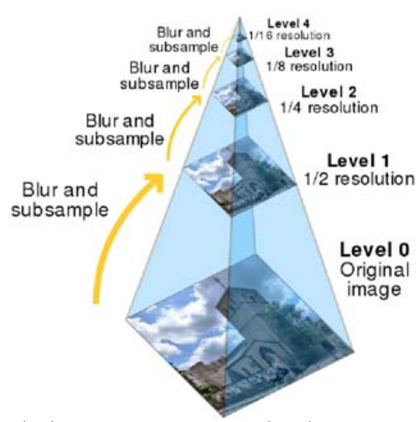
Training Difficulties

- Finding Nash equilibrium (each player feels like they are in a local optimum) difficult.
- Mode collapse- the generator starts to produce several copies of the same (good) instance.
- Oscillation between solutions.
- Initially, hard for generator, easy for discriminator.

LAPGAN

Laplacian Pyramid GAN, Denton et al. (2015)
https://arxiv.org/abs/1506.05751

- GAN creates discriminator and generator.
- After training, discard discriminator
- Generator creates natural images up to 64×64 pixels.

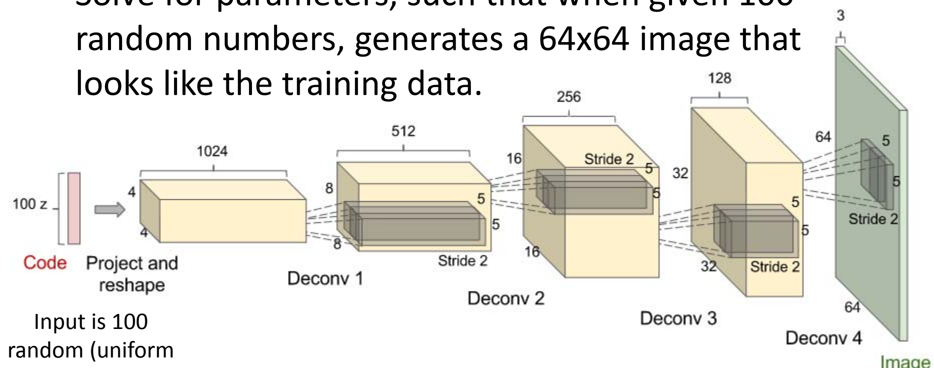


https://en.wikipedia.org/wiki/Pyramid_%28image_processing%29#/media/File:Image_pyramid.svg

DCGAN

Deep Convolutional GAN, Radford et al. (2016)

- Fully convolutional deep CNN generator.
- Solve for parameters, such that when given 100 random numbers, generates a 64x64 image that looks like the training data.



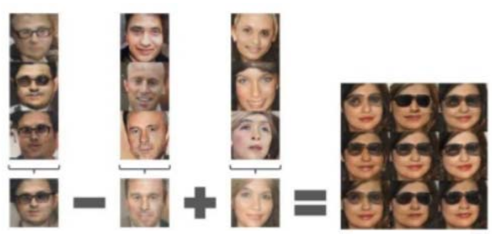
Input is 100 random (uniform distribution) numbers

https://arxiv.org/abs/1511.06434

Output is 64×64×3 image

23

DCGAN- Vector Arithmetic



24

Improved Techniques for Training GANs, Salimans et al. (2016)

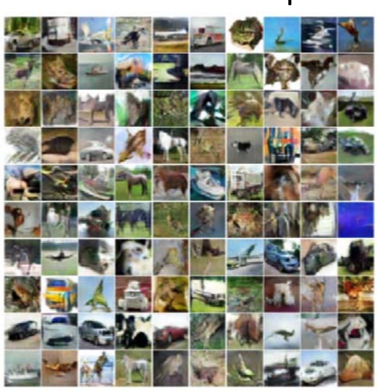
https://arxiv.org/abs/1606.03498

- Techniques to encourage convergence.
- Very difficult as cost functions are not convex, parameters are continuous, and parameter space is high dimensional.
- Modified cost function to encourage better generator.
- Minibatch discrimination to avoid mode collapse:
  - Compute feature statistics across the entire minibatch (not only from individual images). Images generated from each minibatch will exhibit similar statistics.
- Replace batch norm with virtual batch normalization (which uses a reference batch).

Improved Techniques for Training GANs, Salimans et al. (2016)

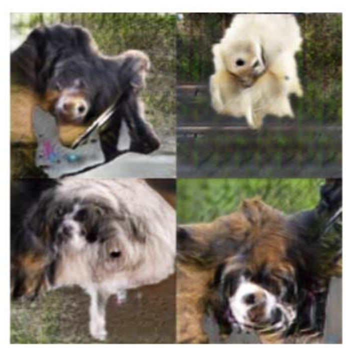
https://arxiv.org/abs/1606.03498

- Use human judges to access quality as well as introduce inception score.



CIFAR10 generated images

SVHN generated images



ImageNet generated images (first to generate 128×128 images, but note low quality due to large number of classes

Improved Techniques for Training GANs, Salimans et al. (2016)

https://arxiv.org/abs/1606.03498

- Also introduced a method to generate a label along with the generated image.
- This is useful for data augmentation on datasets with few samples.
- For example, they achieved 99.14% MNIST accuracy with only 10 labeled examples per class (most approaches use 60K training samples)

InfoGAN
Chen et al. (2016)

https://arxiv.org/abs/1606.03657

- Steering or encourage generator to learn specific representations
- Disentangled representations (facial expression, eye color, hairstyle, glasses, …)
- Uses noise, z and latent code c.

InfoGAN
Chen et al. (2016)

https://arxiv.org/abs/1606.03657

10 different variations- this is done by doing a sweep on the input code to the generator

5 samples

InfoGAN
Chen et al. (2016)

https://arxiv.org/abs/1606.03657

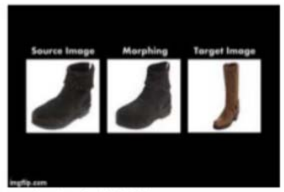
10 different variations- this is done by doing a sweep on the input code to the generator

5 samples

Pose Variations

Lighting Variations

iGAN



Source: Gifs generated from original video (https://www.youtube.com/watch?v=9c4z6YsBGQ0).

cGAN (conditional GAN)

Super resolution

Source to Target Domain Transfer

CycleGAN, Zhu et al., 2017

Generator Evaluation

- GANs create generator and discriminator.
- After training, discriminator often thrown away.
- How do you know which resulting generator implementation is best?

Is this a good image generator output?

How about this?

Generator Evaluation

- Can ask humans to evaluate and rank output from several methods.
- Salimans et al. introduced the concept of an inception score.

Improved Techniques for Training GANs

Tim Salimans tim@openai.com
Ian Goodfellow ian@openai.com
Wojciech Zaremba woj@openai.com
Vicki Cheung vicki@openai.com
Alec Radford alec.radford@gmail.com
Xi Chen peter@openai.com

.G] 10 Jun 2016

Abstract

We present a variety of new architectural features and training procedures that we apply to the generative adversarial networks (GANs) framework. We focus on two applications of GANs: semi-supervised learning, and the generation of images

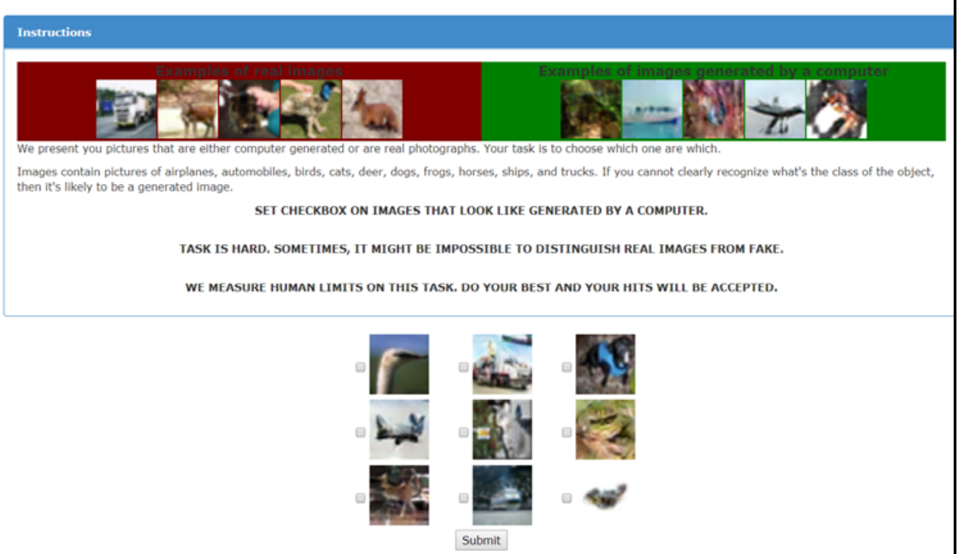
https://arxiv.org/pdf/1606.03498.pdf

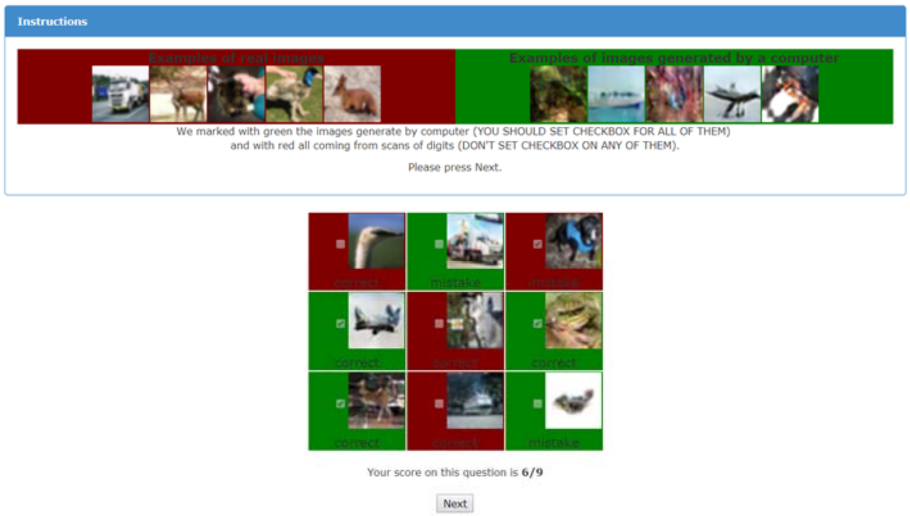
Code: https://github.com/openai/improved-gan/tree/master/inception_score

Assessment of Image Quality of Generated Images

- With no specific objective function which measures image quality, we need to resort to other methods.
- The discriminator should ideally be able to tell real from fake images, but it is easily tricked, and it itself is part of what is being trained.
- Human evaluators can look at images and manually label fake from real.
- Salimans'16 used Mturk to evaluate performance.

http://infinite-chamber-35121.herokuapp.com/ cifar-minibatch/





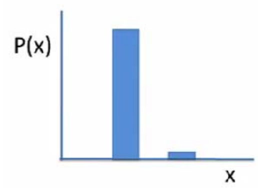
Inception Score

- Use any ImageNet model (such as Google's Inception) to classify an image.
- For each image, a p(y|x), where x is the image, and y is the class distribution is generated.
  – We expect for real images, p(y|x) will concentrate on a few classes; and for fake images, p(y|x) will be more random.
  – As such, we can look at the entropy of p(y|x)

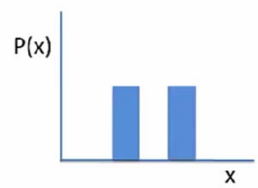
Entropy

$$Entropy = -\sum_{j} p(x_j) log_2 \left(p(x_j)\right)$$

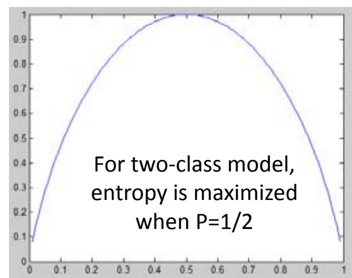
- Entropy is a measure of uncertainty.
- We anticipate P(y|x) will have low entropy for real images.
- We anticipate P(y|x) will have high entropy for fake images.



Low Entropy



High Entropy



KL Divergence

$$D_{KL}(p||q) = -\sum_{j} p(x_j) \left(log_2 \left(p(x_j)\right) - log_2 \left(q(x_j)\right)\right)$$

- Kullback-Leibler (KL) divergence is a measure of how one probability distribution differs from a second.
- For example, given an actual distribution, and an estimate, KL divergence will measure the information lost by using the estimate.
- Generally, you are better off using the distribution with the least information lost.

Code: https://github.com/openai/improved-gan/tree/master/inception_score

Should be n×C , where n is # samples, C is # classes

```
preds = np.concatenate(preds, 0)
scores = []
for i in range(splits):
    part = preds[(i * preds.shape[0] // splits):((i + 1) * preds.shape[0] // splits), :]
    kl = part * (np.log(part) - np.log(np.expand_dims(np.mean(part, 0), 0)))
    kl = np.mean(np.sum(kl, 1))
    scores.append(np.exp(kl))
return np.mean(scores), np.std(scores)
```

For i=0:9

Extract 1/10th of preds at a time
note: a//b is int(a/b)

Part*(log(part) – log(mean(part)))
Mean is done over all samples, one mean for each class, so get 1×C, which is expanded to (n/10)×C
Kl is (n/10)×C

scores will be n×1

mean(sum(kl)), mean across classes
Kl is (n/10)×1

http://anhnguyen.me/project/ppgn/

Plug & Play Generative Networks:
Conditional Iterative Generation of Images in Latent Space

Anh Nguyen
University of Wyoming[†]
anh.ng8@gmail.com

Jeff Clune
Uber AI Labs[†], University of Wyoming
jeffclune@uwyo.edu

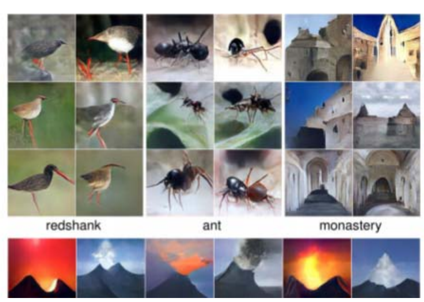
Yoshua Bengio
Montreal Institute for Learning Algorithms
yoshua.umontreal@gmail.com

Alexey Dosovitskiy
University of Freiburg
dosovits@cs.uni-freiburg.de

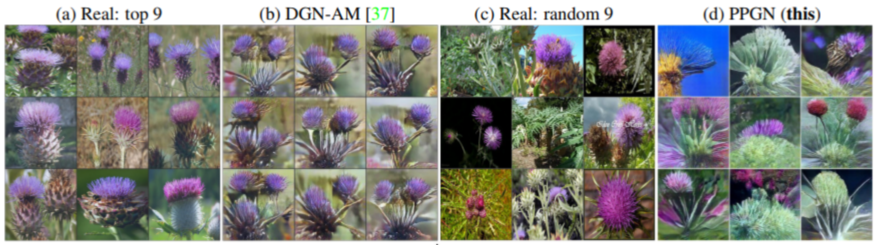
Jason Yosinski
Uber AI Labs[†]
yosinski@uber.com

Abstract

Generating high-resolution, photo-realistic images has been a long-standing goal in machine learning. Recently, Nguyen et al. [37] showed one interesting way to synthesize novel images by performing gradient ascent in the latent space of a generator network to maximize the activations of one or multiple neurons in a separate classifier network.



Class “cardoon” from ImageNet



https://arxiv.org/pdf/1612.00005v2.pdf

The top 9 images from ImageNet that maximally activate the “cardoon” class.

DGN-AM; a generative method which is tasked to generate “cardoon” class.

9 random “cardoon” images from ImageNet.

PPGN; this paper’s generative method induces more diversity.

General Math

Metropolis-adjusted Langevin algorithm converted to a Markov chain Monte Carlo update rule:

$$x_{t+1} = x_t + \epsilon_{12} \nabla \log p(x_t) + N(0, \epsilon_3^2) \qquad (1)$$

x_t input image
x_{t+1} output image
$p(x_t)$ prob that x_t looks like a real image (softmax output).

Normal gradient ascent to generate new images, but add a noise term for randomness.

Update rule used in this paper:

$$x_{t+1} = x_t + \underbrace{\epsilon_1 \frac{\partial \log p(x_t)}{\partial x_t}}_{} + \underbrace{\epsilon_2 \frac{\partial \log p(y = y_c | x_t)}{\partial x_t}}_{} + \underbrace{N(0, \epsilon_3^2)}_{} \qquad (5)$$

Modified (1) for signal generation purposes.

Take a step from x_t to x_{t+1} in a direction such that x_{t+1} looks more like a real image (from any class).

Take a step from x_t to x_{t+1} in a direction such that x_{t+1} looks more like a real image from class y_c.

Add noise to encourage diversity in generated image.

Sampler Update Rule: *p(x)*

$$x_{t+1} = x_t + \epsilon_1 \frac{\partial \log p(x_t)}{\partial x_t} + \epsilon_2 \frac{\partial \log p(y = y_c | x_t)}{\partial x_t} + N(0, \epsilon_3^2) \quad (5)$$

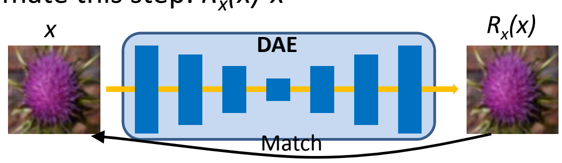
Modified (1) for signal generation purposes.

Take a step from x_t to x_{t+1} in a direction such that x_{t+1} looks more like a real image (from any class).

Take a step from x_t to x_{t+1} in a direction such that x_{t+1} looks more like a real image from class y_c.

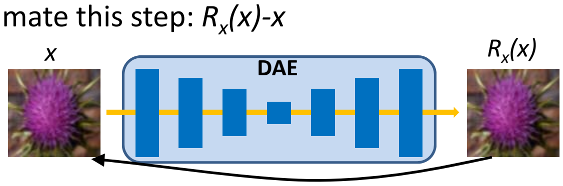
Add noise to encourage diversity in generated image.

- Probability image looks like a real image (from any class)
- Theorized that a Denoising Autoencoder (DAE) can approximate this step: $R_x(x)-x$



Sampler Update Rule: *p(x)*

- Probability image looks like a real image (from any class)
- Theorized that a Denoising Autoencoder (DAE) can approximate this step: $R_x(x)-x$



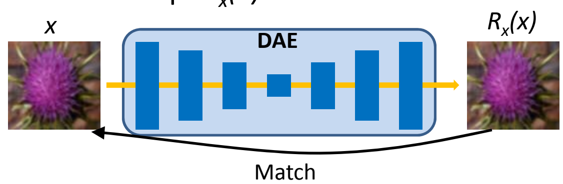
Match

Reduced dim space, just like bottleneck of DAE.

- Modeling DAE in image space has two problems:
  1. Does not model distribution accurate enough
  2. Small update contributions
- Improvements suggest to first encode image to *h*, some image2vec encoded space (say fc6 from AlexNet).

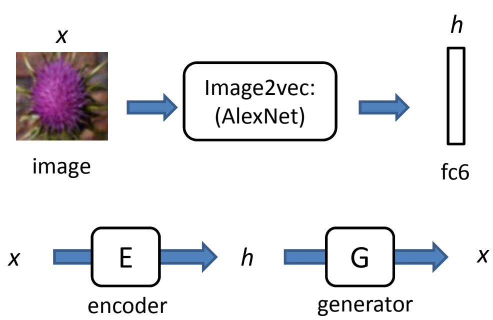
Sampler Update Rule: *p(x)*

- Probability image looks like a real image (from any class)
- Theorized that a Denoising Autoencoder (DAE) can approximate this step: $R_x(x)$-x



- Improvements suggest to first encode image to *h*, some image2vec encoded space (say fc6 from AlexNet).
- Empirical studies found that this did not work too well either, but modeling *h* via *x* seems to help.

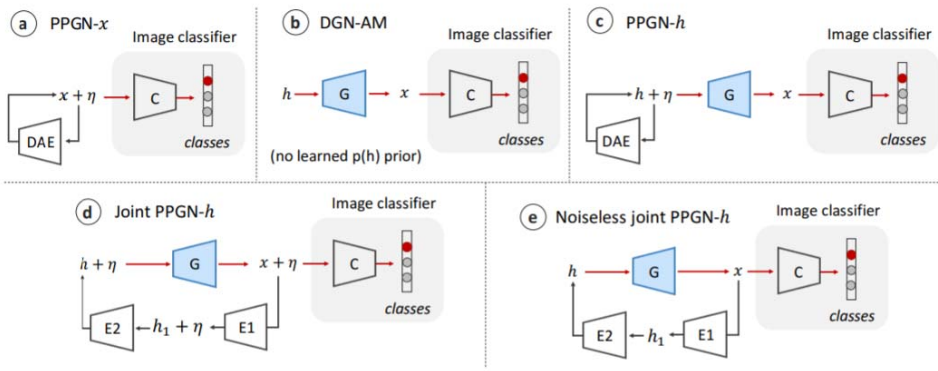
Sampler Update Rule: *p(x)*



Hmm…our task is to generate good looking images via a generator!

- Suggest to first encode image to *h*, some image2vec encoded space (say fc6 from AlexNet), then back to *x* (use generator).
- Empirical studies found that this did not work too well either, but modeling *h* via *x* seems to help.
- Try updating in *h* instead of *x*!

Several DAE Experiments

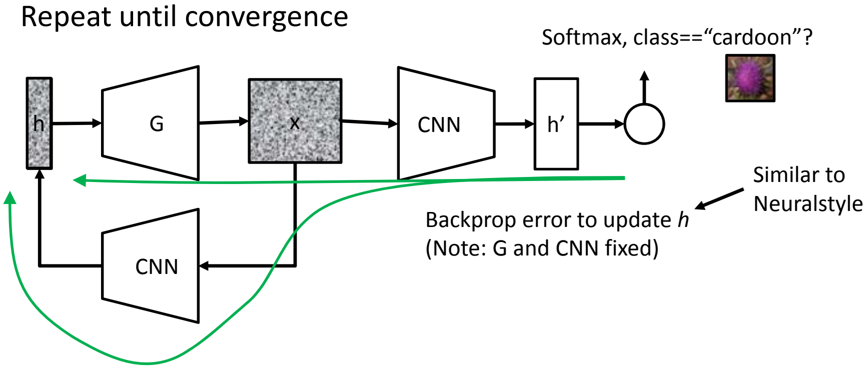


Sampling conditioning on **classes**

Many variants tried!

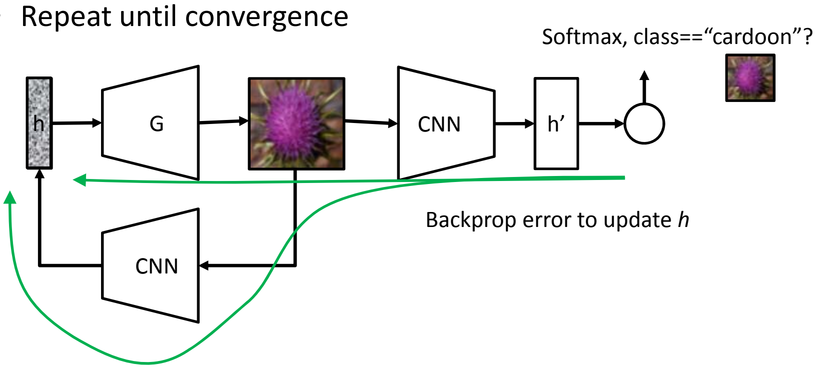
PPGN In Practice: Generate Images for a Certain Class

- Start with random *h*, compare generated class with desired
- Backprop error to *h* (fixed G and CNN), regenerate class,
- Repeat until convergence



PPGN In Practice: Generate Images for a Certain Class

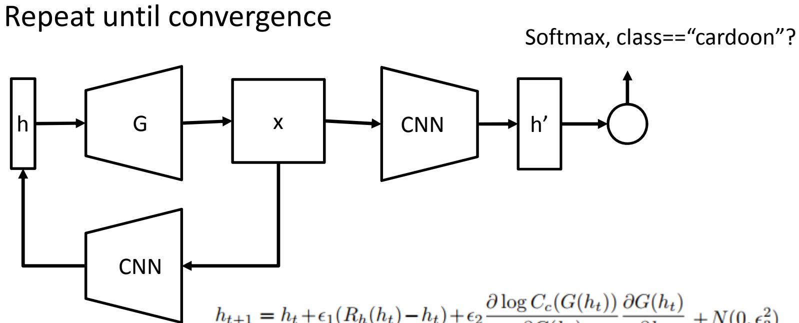
- Start with random *h*, compare generated class with desired
- Backprop error to *h* (fixed G and CNN), regenerate class,
- Repeat until convergence



53

PPGN In Practice: Generate Images for a Certain Class

- Start with random *h*, compare generated class with desired
- Backprop error to *h* (fixed G and CNN), regenerate class,
- Repeat until convergence



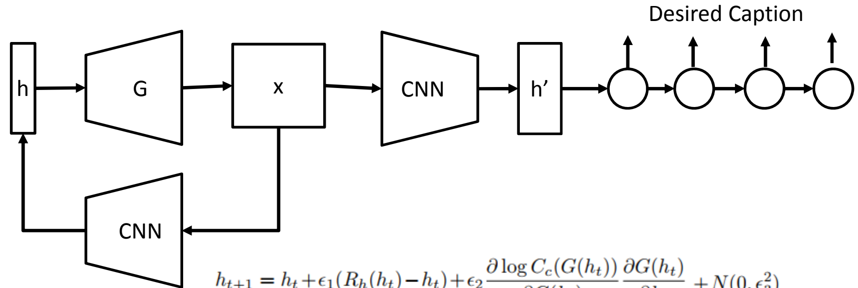
$$h_{t+1} = h_t + \epsilon_1 (R_h(h_t) - h_t) + \epsilon_2 \frac{\partial \log C_c(G(h_t))}{\partial G(h_t)} \frac{\partial G(h_t)}{\partial h_t} + N(0, \epsilon_3^2)$$

Pixel reconstruction error | Cross-entropy loss | Random noise

54

PPGN In Practice: Generate Images from a Caption

- Start with random h, compare generated caption with desired
- Backprop error to h (fixed G and CNN), regenerate caption,
- Repeat until convergence

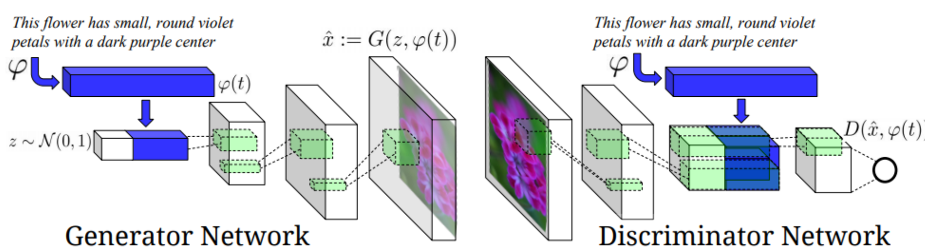


$$h_{t+1} = h_t + \epsilon_1 (R_h(h_t) - h_t) + \epsilon_2 \frac{\partial \log C_c(G(h_t))}{\partial G(h_t)} \frac{\partial G(h_t)}{\partial h_t} + N(0, \epsilon_3^2)$$

Pixel reconstruction error | Caption loss | Random noise

Alternate Method using GANS for Text2Image

Reed, Scott, et al. "Generative adversarial text to image synthesis." *arXiv preprint arXiv:1605.05396*(2016).

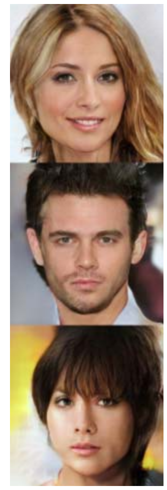


http://research.nvidia.com/sites/default/files/pubs/2017-10_Progressive-Growing-of/karras2018iclr-paper.pdf

Published as a conference paper at ICLR 2018

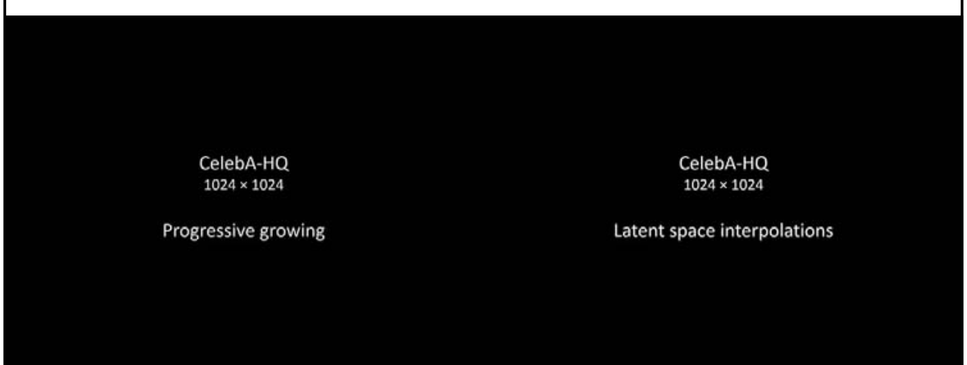
Progressive Growing of GANs for Improved Quality, Stability, and Variation

Tero Karras NVIDIA
Timo Aila NVIDIA
Samuli Laine NVIDIA
Jaakko Lehtinen NVIDIA and Aalto University
{tkarras,taila,slaine,jlehtinen}@nvidia.com



Abstract

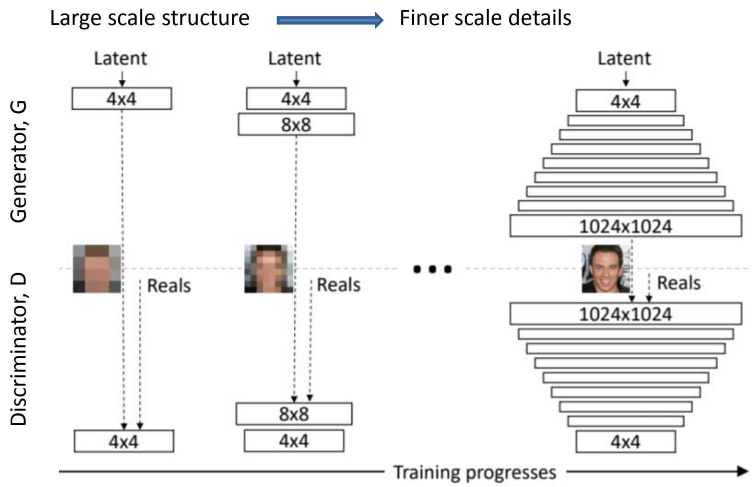
We describe a new training methodology for generative adversarial networks. The key idea is to grow both the generator and discriminator progressively: starting from a low resolution, we add new layers that model increasingly fine details as training progresses. This both speeds the training up and greatly stabilizes it, allowing us to produce images of unprecedented quality, e.g., CELEBA images at 1024^2. We also propose a simple way to increase the variation in generated images, and achieve a record inception score of 8.80 in unsupervised CIFAR10. Additionally, we describe several implementation details that are important for discouraging unhealthy competition between the generator and discriminator. Finally, we suggest a new metric for evaluating GAN results, both in terms of image quality and variation. As an additional contribution, we construct a higher-quality version of the CELEBA dataset.



http://research.nvidia.com/sites/default/files/pubs/2017-10_Progressive-Growing-of/karras2017gan-paper.pdf

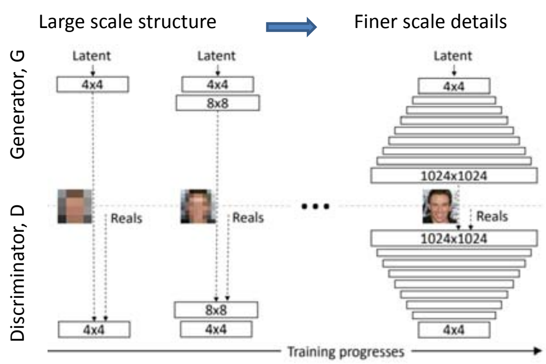
Progressing Growing of GANs

Start with low resolution, and then incrementally increase resolution.



 59

Progressing Growing of GANs

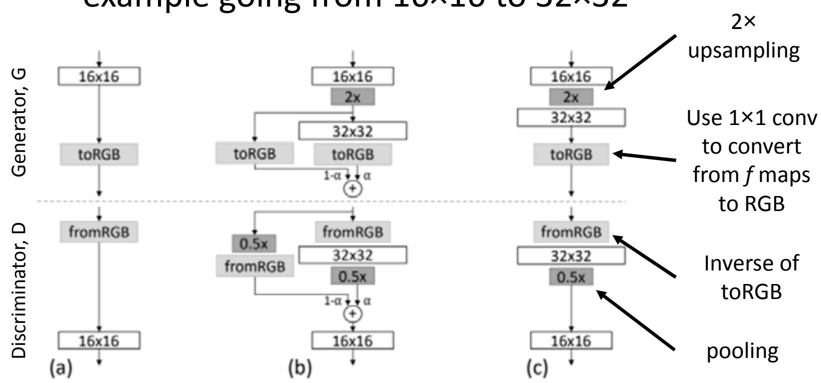


- G and D are mirror architectures, and both trained throughout the process.
- When done, G is used to create images from latent $h \in \mathbb{R}^{512}$

 60

Feathering from Lower to Higher Res

example going from 16×16 to 32×32



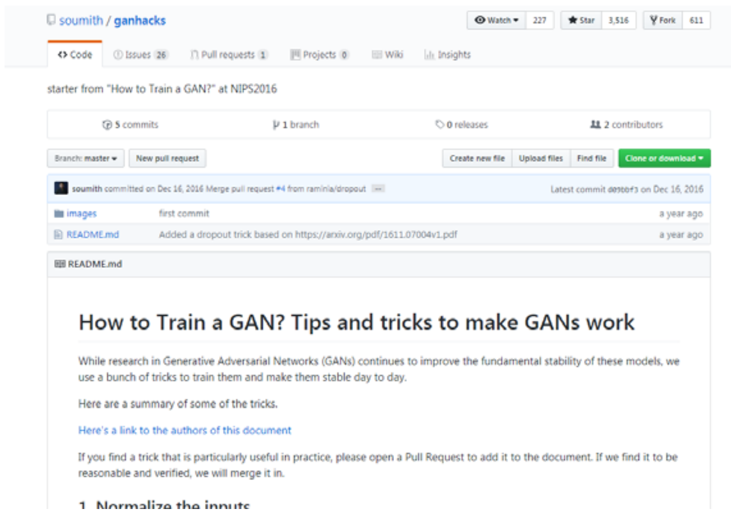
- During the transition (b), layers that operate on the higher resolution like a residual block, increase using weight α linearly from 0 to 1 over time.
- When training the discriminator, real images are downscaled to match the current resolution of the network.

Normalization

- Unhealthy competition between networks G and D are prone to the escalation of signal magnitudes.
- Prior works use batch normalization in the generator, and often also in the discriminator.
- This paper did not observe much covariance shift in GANs, and concludes batchnorm's benefit is really constraining signal magnitudes during training. Instead, this paper used:
  – Equalized Learning Rate- simple variation of He 2015 Xiavier for ReLU applied at runtime.
  – Pixelwise normalization in generator- similar to local response normalization (Krizhevsky, 2012).

Training your own GAN?

https://github.com/soumith/ganhacks



 63

Thank you!!

Ray Ptucha

rwpeec@rit.edu

https://www.rit.edu/mil

 64