Generative Adversarial Networks

Yuan YAO

HKUST

Based on Feifei Li, Aleksander Mądry, Tong Zhang, et al.

Overview

- Unsupervised Learning vs. Supervised Learning
- Generative Models
  - Variational Autoencoders (VAE)
  - **Generative Adversarial Networks (GAN)**

Generative Adversarial Networks (GAN)

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

VAEs define intractable density function with latent **z**:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: don't work with any explicit density function!
Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

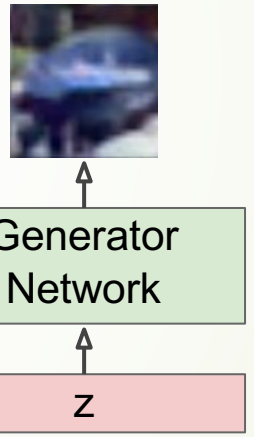
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!

Output: a sample from training distribution



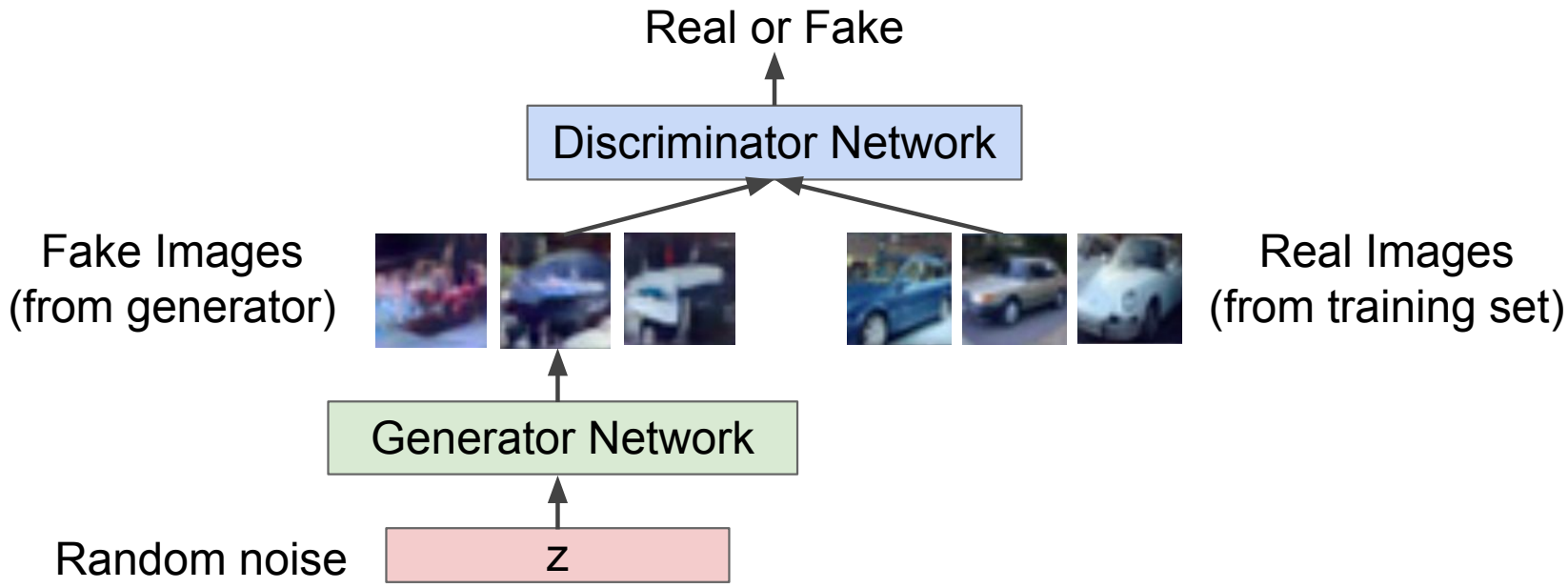
Input: Random noise

How to train?

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



(Original) GANs

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

A minimax game with different objective functions:

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

(Original) GANs

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminatortwork: try to distinguish between real and fake images

Different Objective functions in a Minimax Game:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

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

For training, alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Note: Oscillations might happen…

Training GANs: An Issue

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

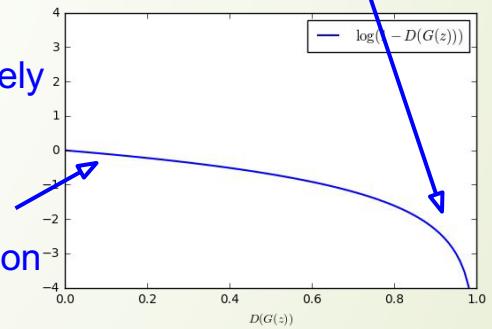
2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GANs: the Log D trick

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

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

Alternate between:

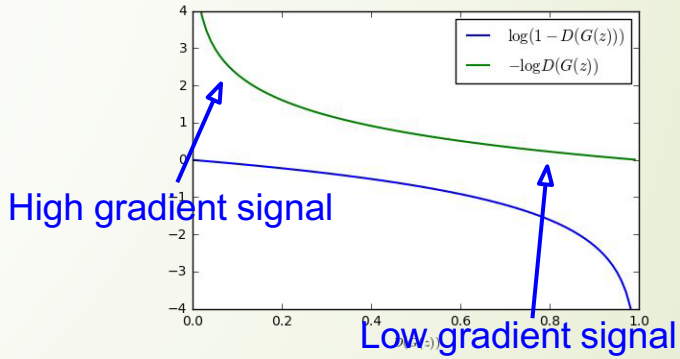
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
now higher gradient signal for bad samples => works much better!
Standard in practice.



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Putting it together: GAN training algorithm

Some find k=1 more stable, others use k > 1 (e.g. 20), no best rule.

for number of training iterations **do**

  for k steps **do**

  - 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:

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

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - 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

Other Losses (Wasserstein Distance, KL-divergence) are better in stability!

Wasserstein GAN [Arjovsky-Chintala-Bottou'17]

- Think F: true high-dimensional distribution (e.g. faces) in $\mathbb{R}^n$
- Q: output of a Deep NN G, of certain architecture, with parameters θ_g
  - i.e. $G_{\theta_g}(z)$, where $z \sim N(0, I)$
- Suppose interested in Wasserstein distance:

$$W(F, Q) = \inf_{\gamma \in \Gamma(F,Q)} (\mathbb{E}_{(X,Y)\sim\gamma}[\|X - Y\|_1]) =$$

Couplings of F, Q

- In a perfect w
- In practice, hard to compute $\inf$ over all 1-Lipschitz functions, so only take sup over all Deep NNs D_w of certain architecture, w/ parameters w:

$$\inf_{\theta} \sup_{w} (\mathbb{E}_{X\sim F}[D_w(X)] - \mathbb{E}_{z\sim N(0,I)}[D_w(G_\theta(z))])$$

- In other words, set up a *game* between a *Generator* deep NN, and a *Discriminator* deep NN

Wasserstein GAN
[Arjovsky-Chintala-Bottou'17]

- Think F: true high-dimensional distribution (e.g. faces) in $\mathbb{R}^n$
- Q: output of a Deep NN G, of certain architecture, with parameters θ_g
  - i.e. $G_{\theta_g}(z)$, where $z \sim N(0, I)$
- Suppose interested in Wasserstein distance:

$$W(F, Q) = \inf_{\gamma \in \prod(F,Q)} \left(\mathbb{E}_{(X,Y)\sim\gamma}[\|X - Y\|_1]\right) = \sup_{D:\mathbb{R}^n \to \mathbb{R},\ 1-Lipschitz} \left(\mathbb{E}_{X\sim F}[D(X)] - \mathbb{E}_{X\sim Q}\left[D(X)\right]\right)$$

- In a perfect world, G_{θ_g} should minimize:

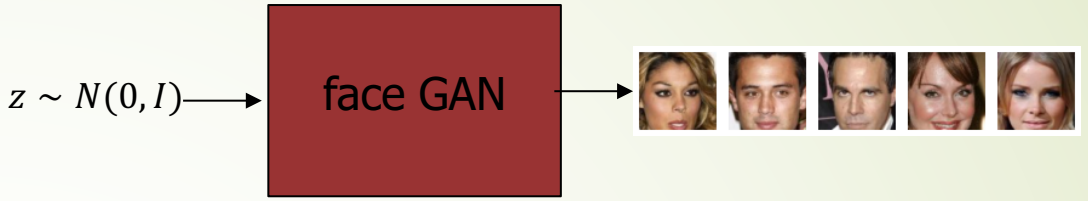
$$\inf_{\theta_g} \sup_{\substack{D:\mathbb{R}^n \to \mathbb{R}, \\ 1-Lipschitz}} \left(\mathbb{E}_{X\sim F}[D(X)] - \mathbb{E}_{Z\sim N(0,I)}\left[D\left(G_{\theta_g}(z)\right)\right]\right)$$

- In practice, hard to compute sup over all Lipschitz functions, so only take sup over all Deep NNs D, of certain architecture, w/ parameters θ_d (clipped for Lipschitzness):

$$\inf_{\theta_g} \sup_{\theta_d} \left(\mathbb{E}_{X\sim F}\left[D_{\theta_d}(X)\right] - \mathbb{E}_{Z\sim N(0,I)}\left[D_{\theta_d}\left(G_{\theta_g}(z)\right)\right]\right)$$

- In other words, set up a ***game*** between a *Generator* deep NN, and a *Discriminator* deep NN

Wasserstein GAN
[Arjovsky-Chintala-Bottou'17]

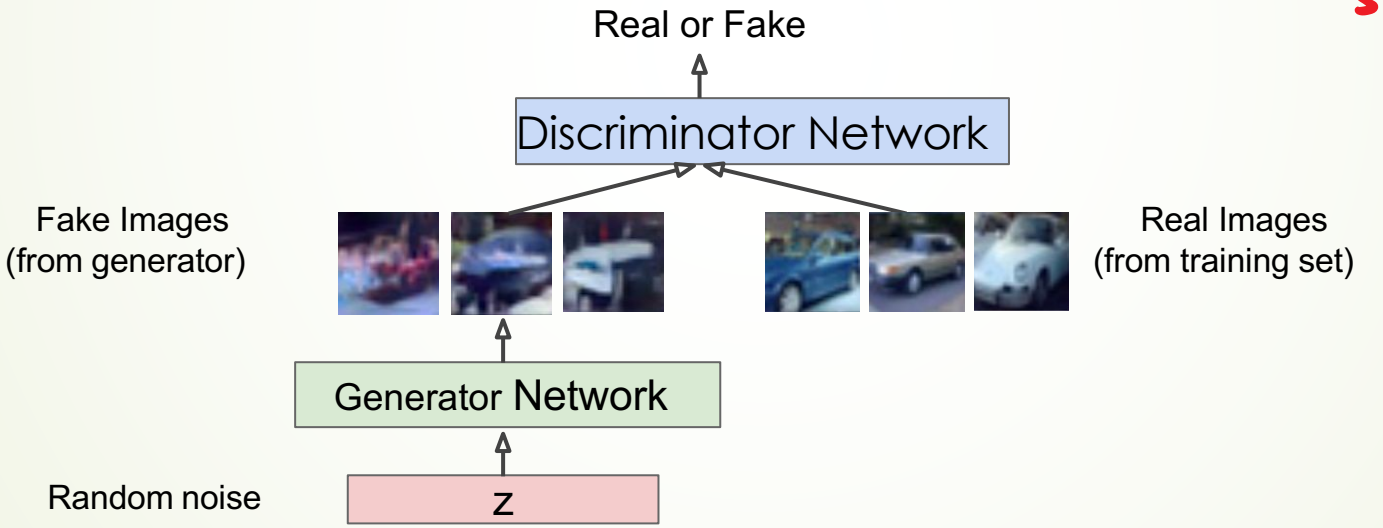


- A ***game*** between a *Generator* deep NN, w/ parameters θ_g, and a *Discriminator* deep NN, w/ parameters θ_d:

$$\inf_{\theta_g} \sup_{\theta_d} \left(\mathbb{E}_{X \sim F}[D_{\theta_d}(X)] - \mathbb{E}_{Z \sim N(0,I)}\left[D_{\theta_d}\left(G_{\theta_g}(Z)\right)\right] \right)$$

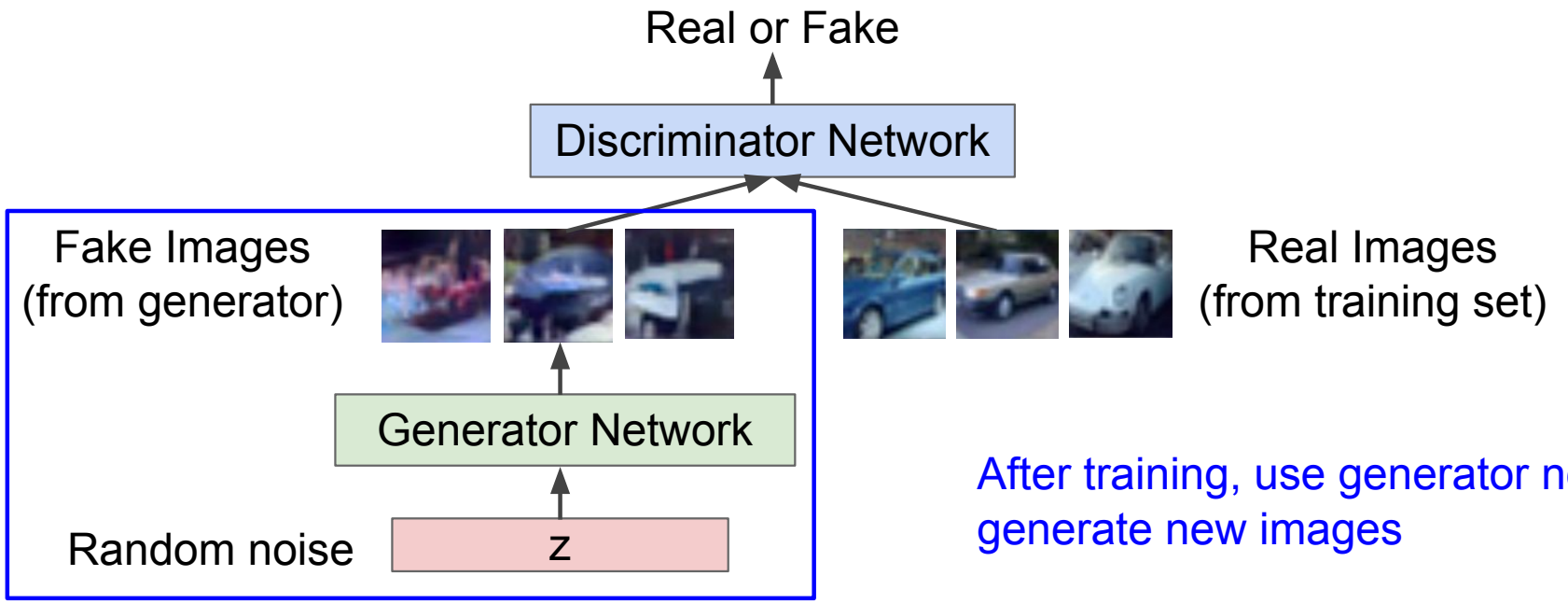
sample from real F

hallucinated sample



- **Training**: generator and discriminator run some variant of gradient descent each to update their parameters θ_g, θ_d; expectations are approximated by finite sample averages
- even ignoring expectation approximation errors, will gradient descent converge? to what?

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

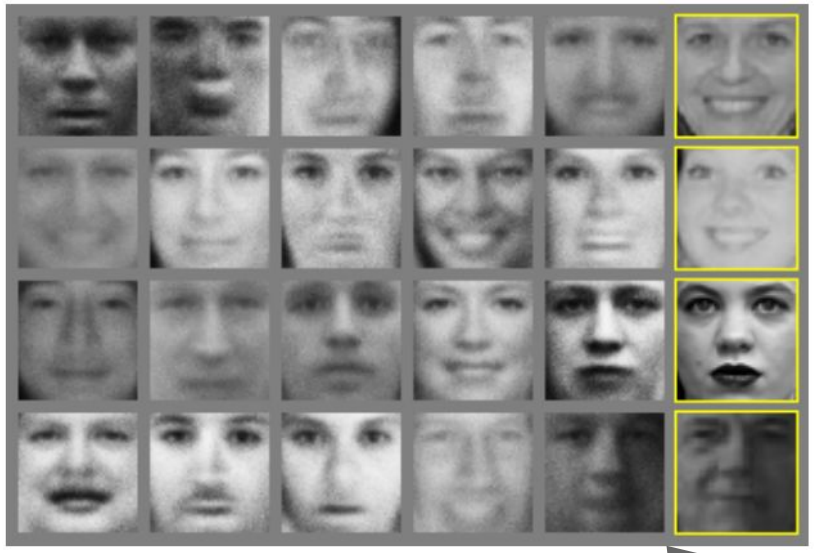


After training, use generator network to generate new images

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generative Adversarial Nets

Generated samples



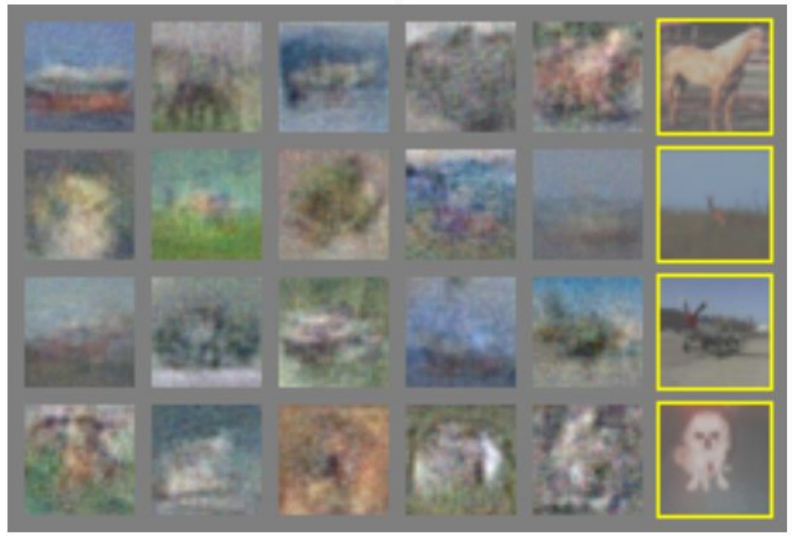
Nearest neighbor from training set

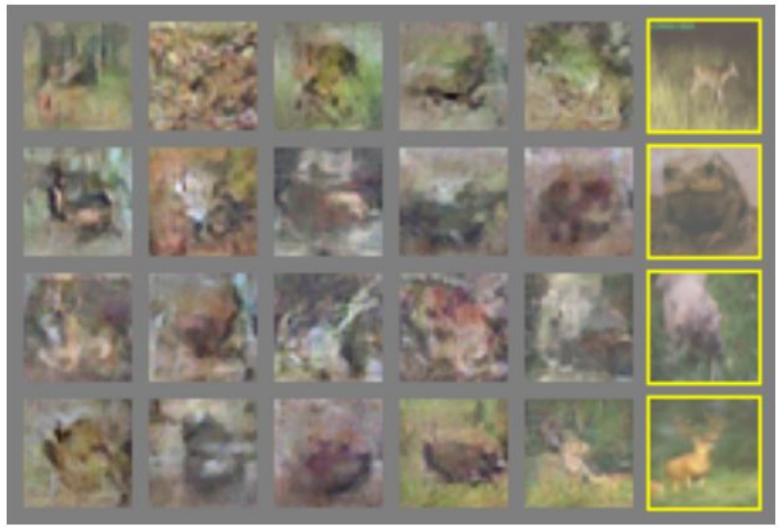
Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generative Adversarial Nets

Generated samples (CIFAR-10)





Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

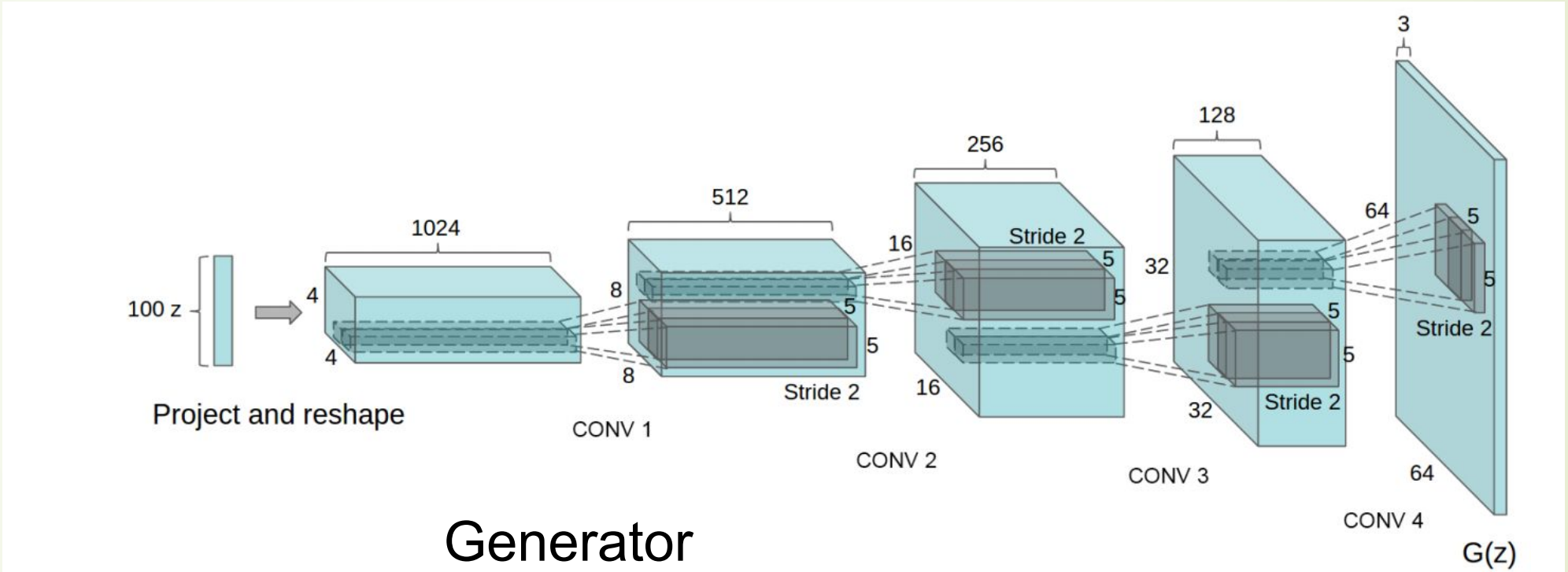
Generative Adversarial Nets: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

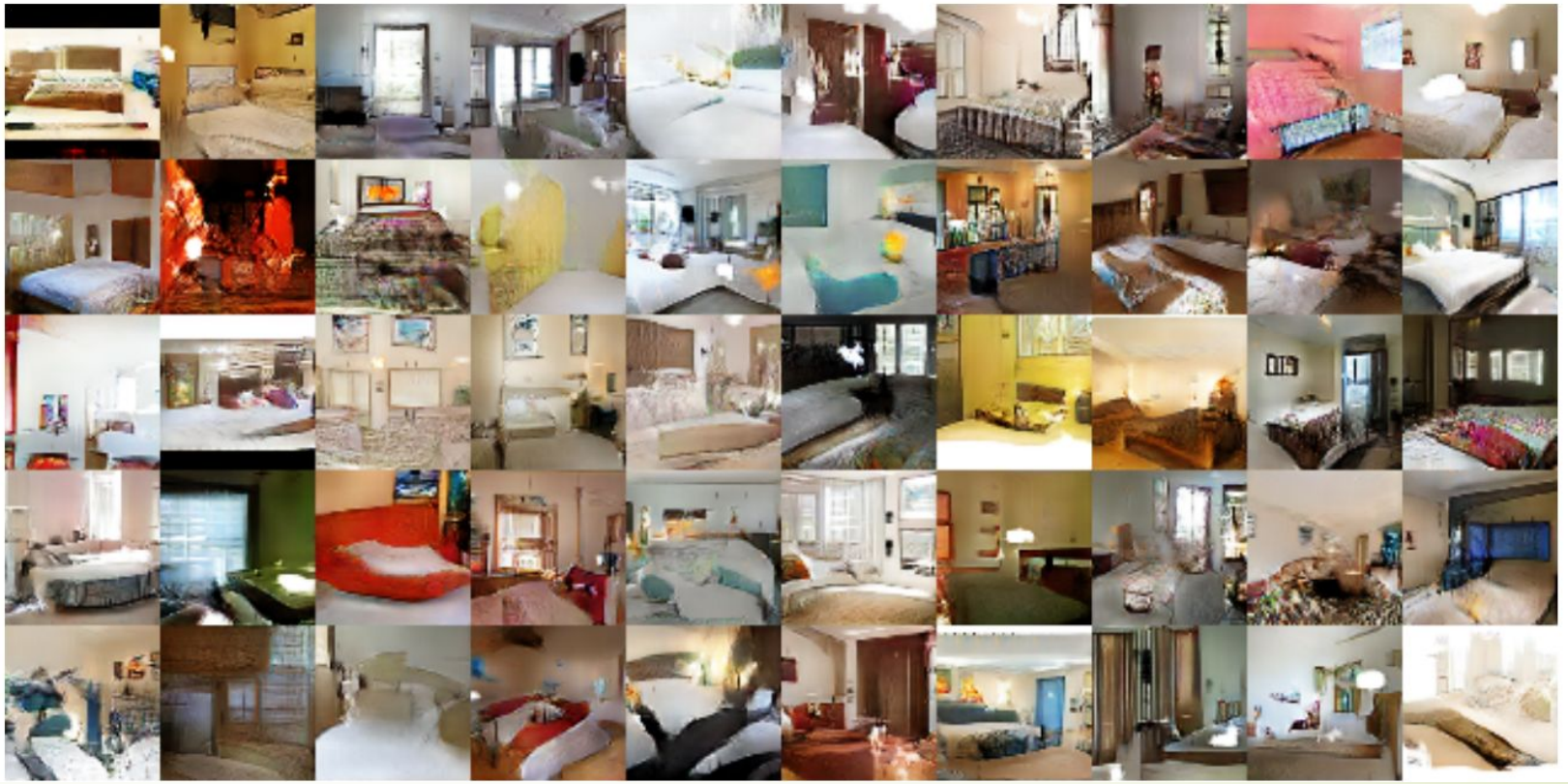


Generator

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

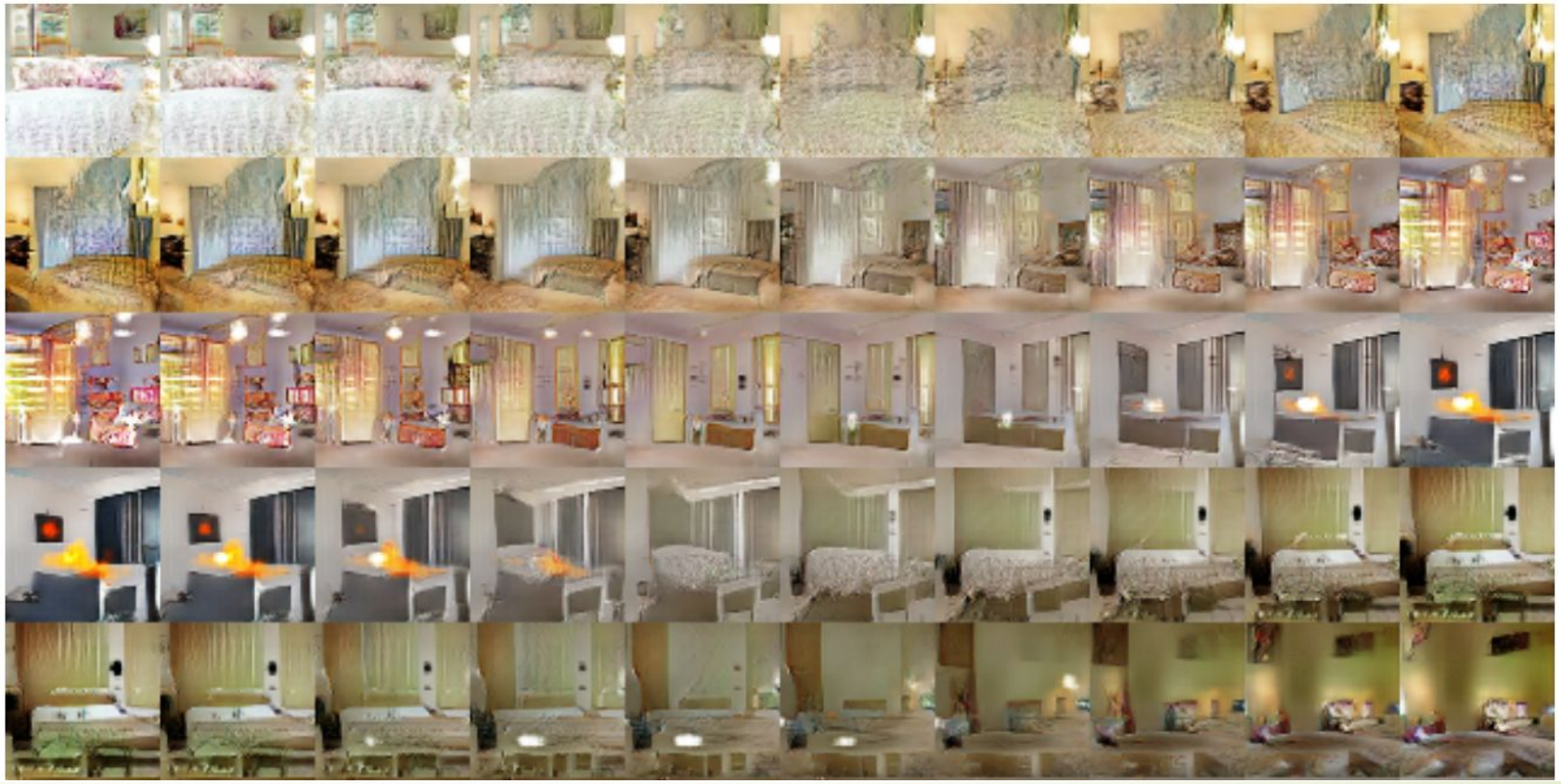
Samples from the model look amazing!



Radford et al, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Interpolating between random points in laten space



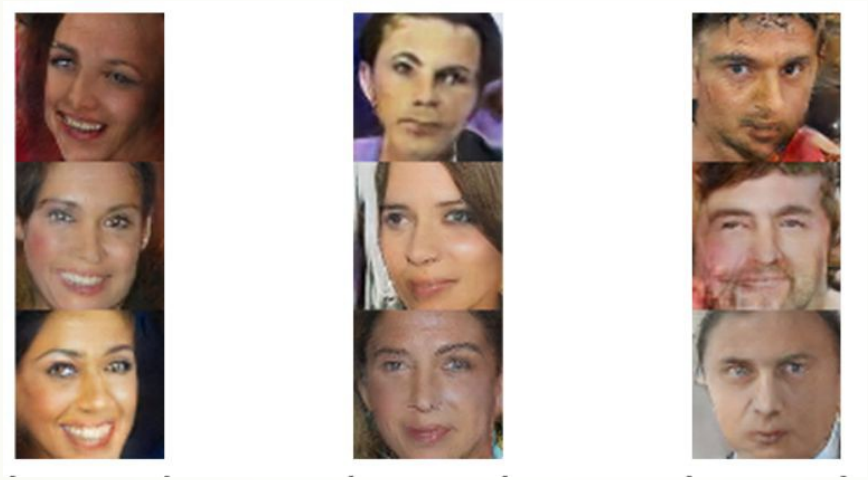
Radford et al, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

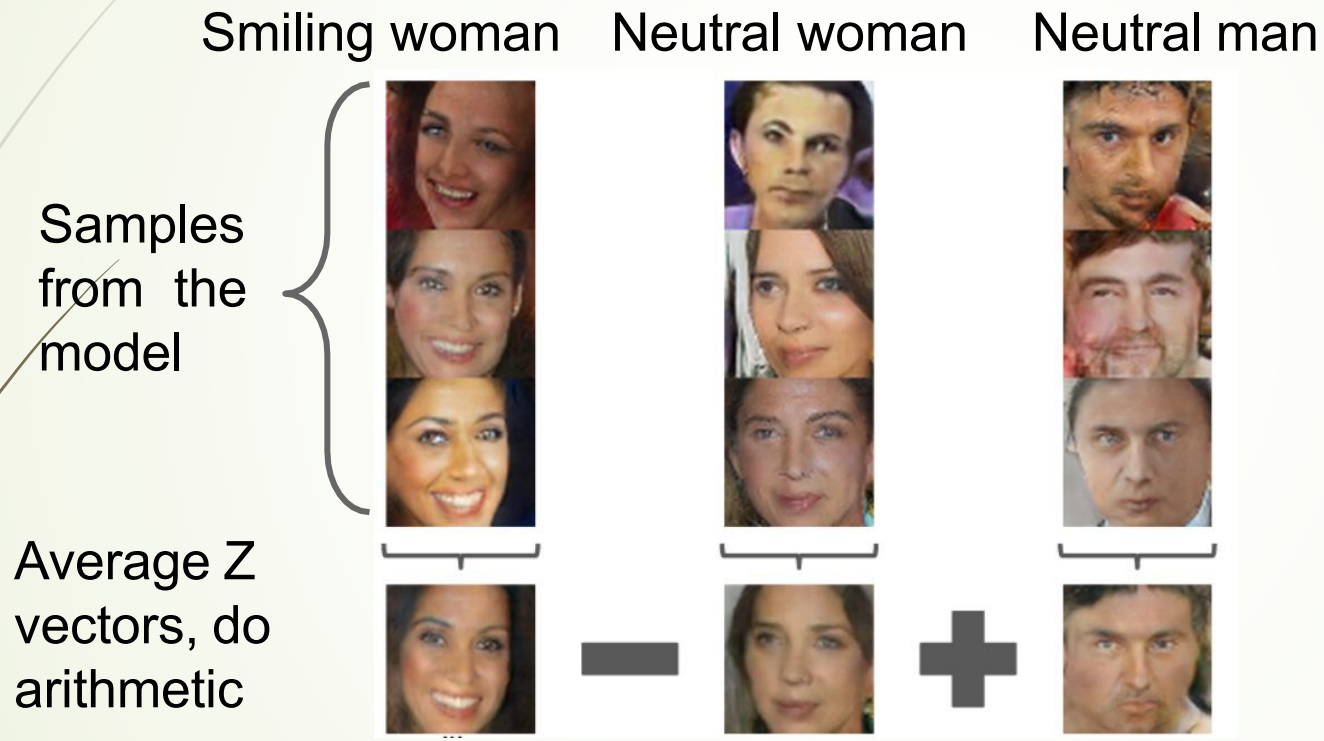
Smiling woman | Neutral woman | Neutral man

Samples from the model



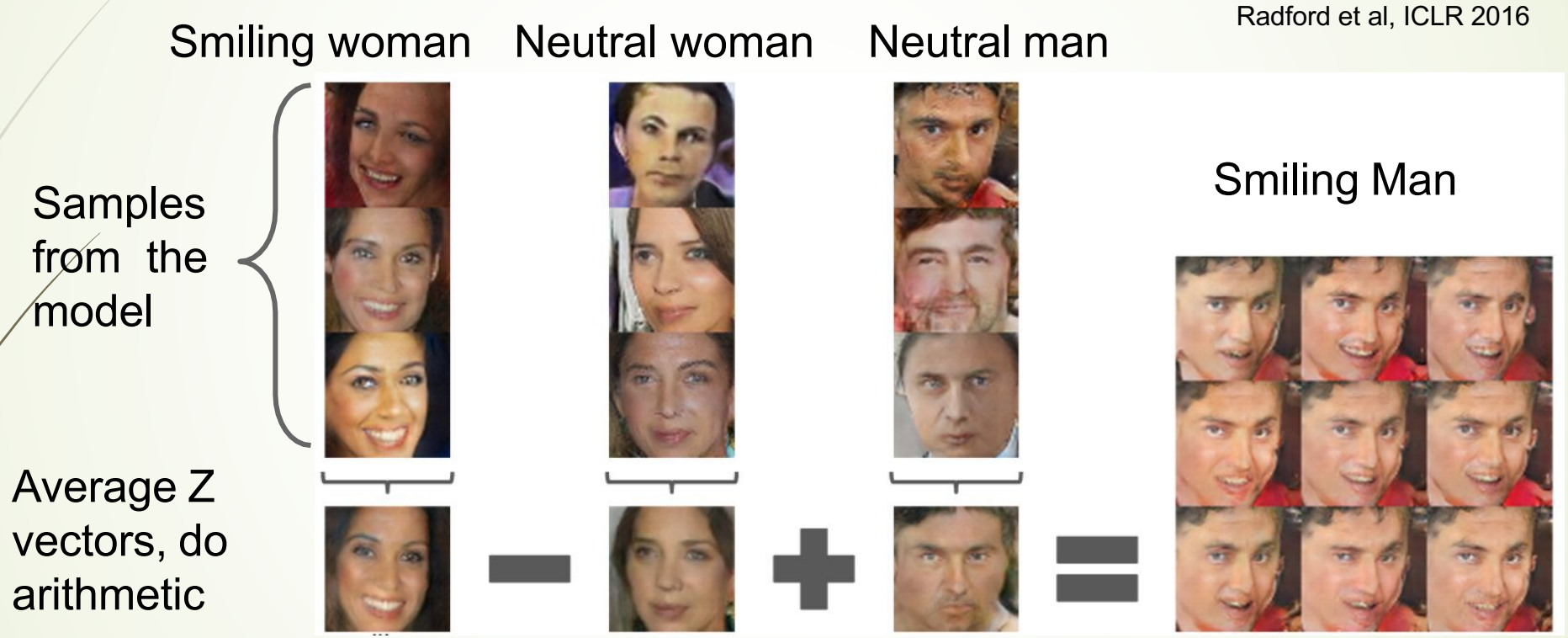
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



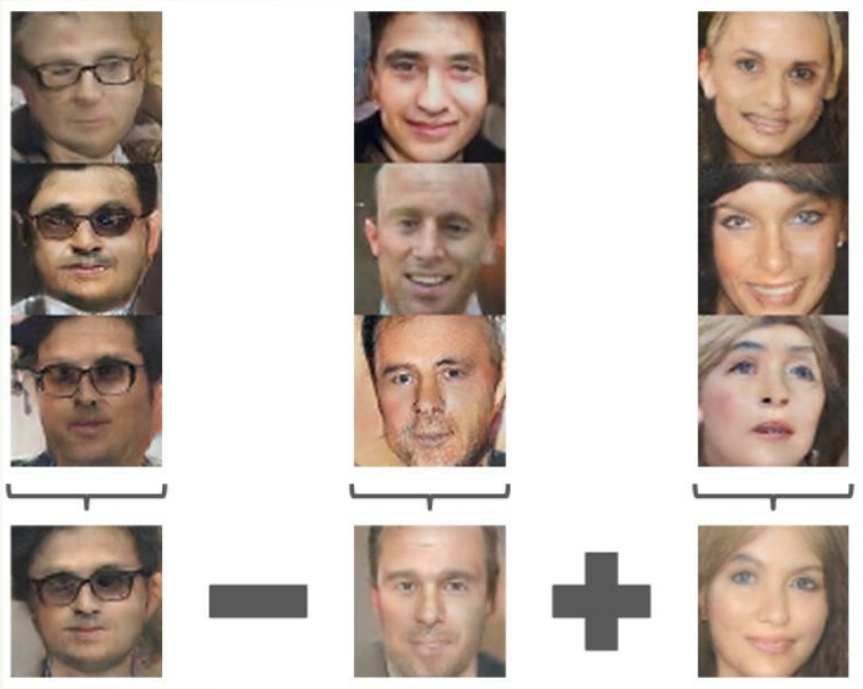
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016



Generative Adversarial Nets: Interpretable Vector Math

Glasses man No glasses man No glasses woman



Radford et al, ICLR 2016

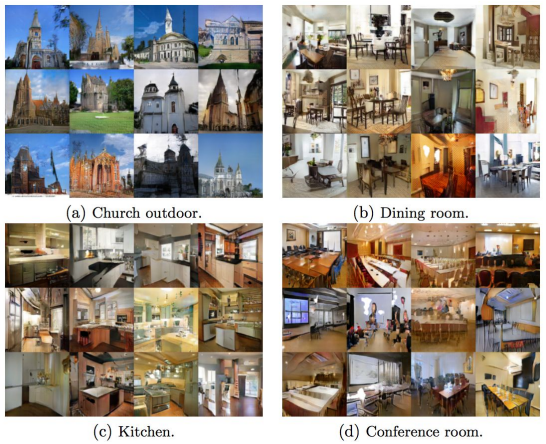
Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Glasses man — No glasses man + No glasses woman = Woman with glasses

2017: Year of the GAN

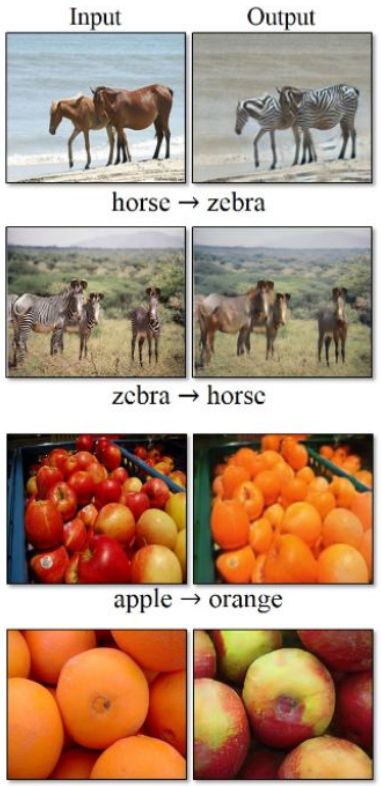
Better training and generation

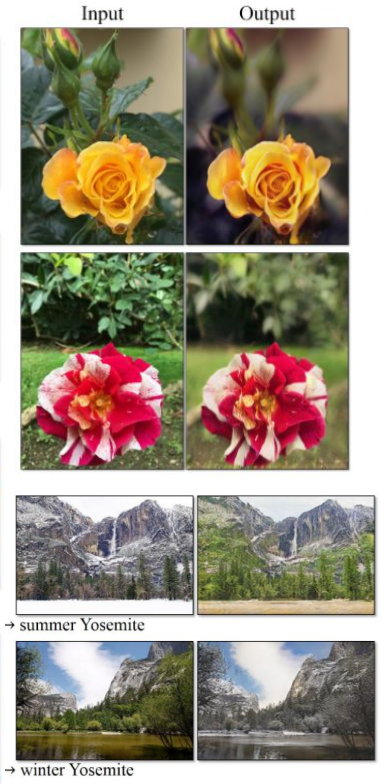


LSGAN. Mao et al. 2017.

BEGAN. Bertholet et al. 2017.

Source->Target domain transfer





CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

Reed et al. 2017.

Many GAN applications

Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

Reference of GANs

- The GAN zoo: https://github.com/hindupuravinash/the-gan-zoo
- See also: https://github.com/soumith/ganhacks for tips and tricks for trainings GANs

GANs

- Don't work with an explicit density function
  Take game-theoretic approach: learn to generate from training distribution through 2-player minimax zero-sum game
- Pros:
  - Beautiful, state-of-the-art samples!
- Cons:
  - Trickier / more unstable to train
  - Can't solve inference queries such as p(x), p(z|x)
- Active areas of research:
  - Better loss functions, more stable training (Wasserstein GAN, LSGAN, etc. Tong ZHANG next time)
  - Conditional GANs, GANs for all kinds of applications

Composite Functional Gradient Learning of Generative Adversarial Models

Rie JOHNSON, *IBM*, and **Tong ZHANG**, *Tencent AI Lab*

Recall that

Want to draw samples similar to $S = \{x_1, \ldots, x_n\}$

Procedure:

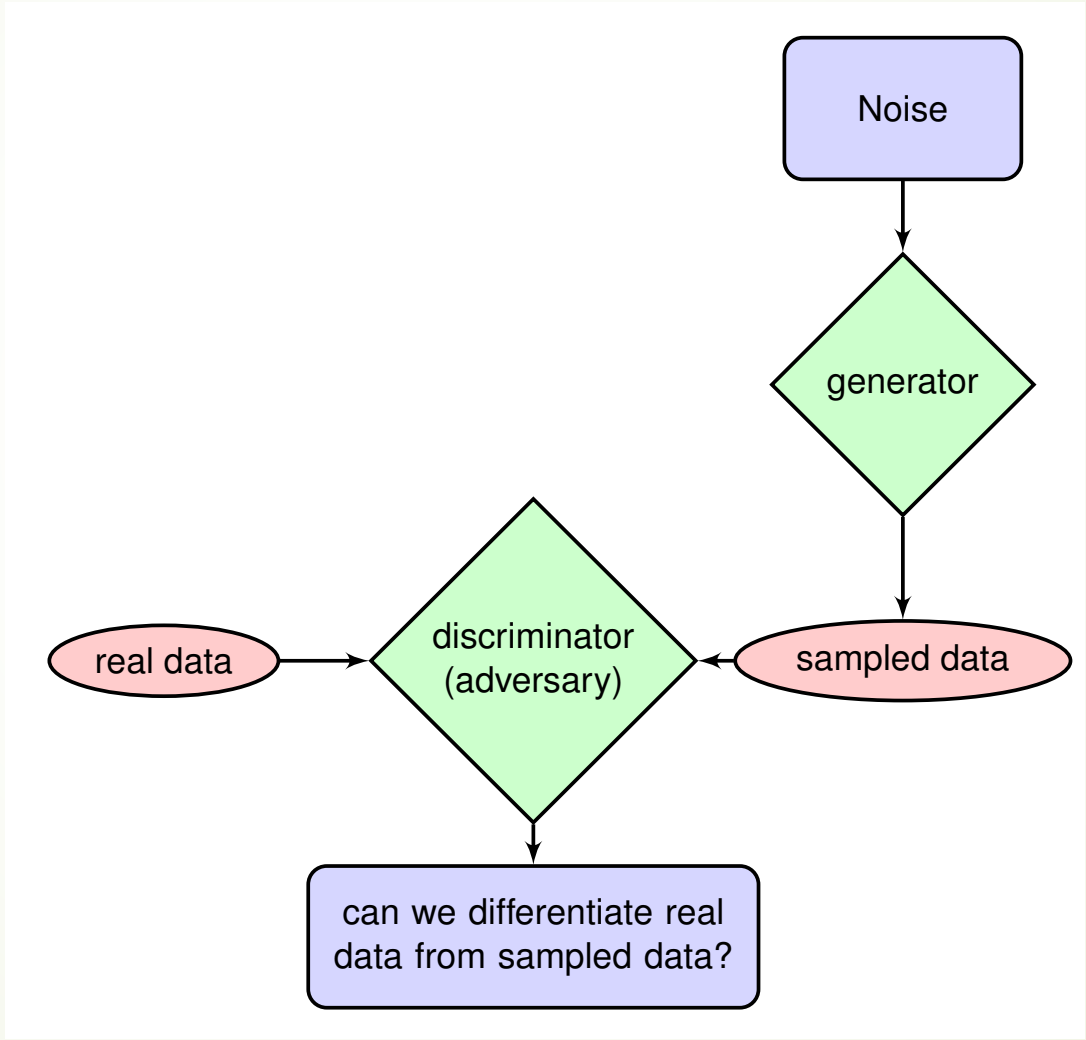
- Generate z from an initial noise distribution p_z
- Transform $z : z \to G(z)$ so that $G(z)$ has the same distribution as S

Key problem:

- How to estimate the transformation $G(z)$?

Modern answers: GAN (and related, VAE)

Generative Adversarial Network (GAN)



Mathematical Formulation of GAN

- Real data $S = \{x_1, \ldots, x_n\}$
- Noise $z_1, \ldots, z_m$
- Generator $G(\theta; z)$
- Discriminator $d(\psi; x)$

Nonconvex Minimax Saddle Point Optimization Problem:

$$\max_{\psi} \min_{\theta} \left[\frac{1}{n} \sum_{i} \log d(\psi; x_i) + \frac{1}{m} \sum_{j} \log(1 - d(\psi; G(\theta; z_j))) \right].$$

We have:

$$d(\psi; x) = \frac{1}{1 + \exp(-D(\psi; x))}$$

Solving Saddle Point Optimization

$$\max_{\psi} \min_{\theta} \left[\frac{1}{n} \sum_{i} \log d(\psi; x_i) + \frac{1}{m} \sum_{j} \log(1 - d(\psi; G(\theta; z_j))) \right].$$

Using SGD as follows.

- Update discriminator (primal variable SGD):

$$\psi \leftarrow \psi + \eta \nabla_{\psi} \left[\log d(\psi; x_i) + \log(1 - d(\psi; G(\theta; z_j))) \right]$$

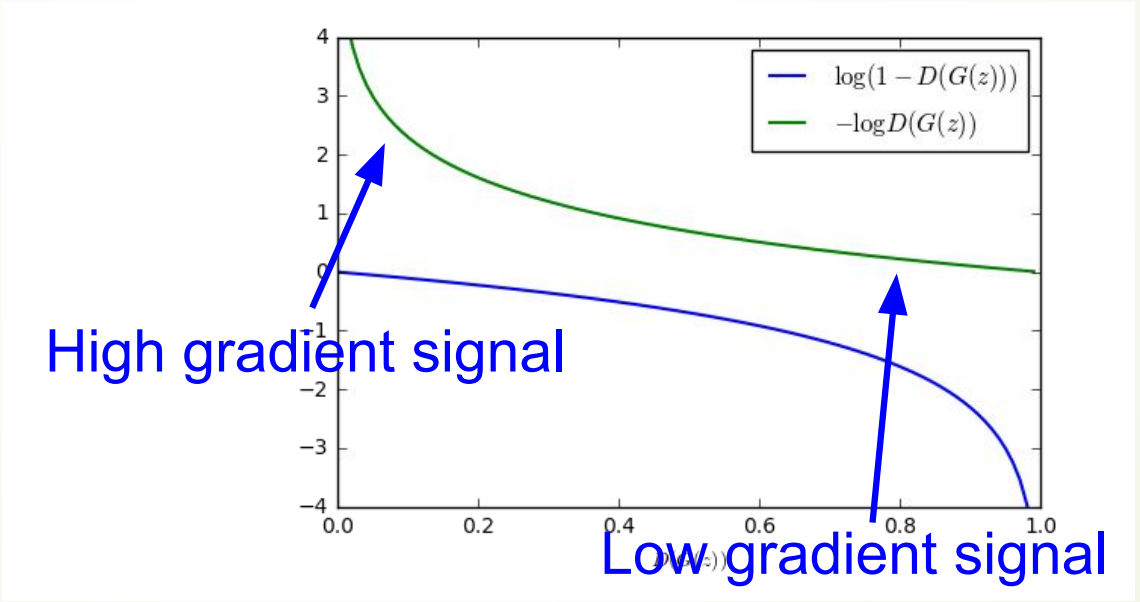
- Update generator (dual variable SGD):

$$\theta \leftarrow \theta - \eta' \nabla_{\theta} \log(1 - d(\psi; G(\theta; z_j)))$$

The Log D trick

Strange phenomenon: generator update with logD trick is practically better

$$\theta \leftarrow \theta + \eta' \nabla_\theta \log(d(\psi; G(\theta; z_j))$$



Mathematical Theory of GAN

The population version of GAN tries to find optimal d to maximize

$$\int \log d(x) p_{\text{real}}(x) dx + \int \log(1 - d(x)) p_{\text{gen}}(x) dx,$$

where

$p_{\text{gen}}(x)$ is the density of $x = G(z)$

The optimal d is given by

$$d_{\text{optimal}}(x) = \frac{p_{\text{real}}(x)}{p_{\text{real}}(x) + p_{\text{gen}}(x)}.$$

Generator Minimizes JS-divergence

$$\max_{d} \min_{G} \left[\mathbf{E}_x \log d(x) + \mathbf{E}_z \log(1 - d(G(z))) \right].$$

Substitute d by d_{optimal}, we get Jensen-Shanon (JS) divergence minimization:

$$\min_{G} \left[\int p_{\text{real}}(x) \log \frac{2p_{\text{real}}(x)}{p_{\text{real}}(x) + p_{\text{gen}}(x)} dx + \int p_{\text{gen}}(x) \log \frac{2p_{\text{gen}}(x)}{p_{\text{real}}(x) + p_{\text{gen}}(x)} dx \right],$$

where

$p_{\text{gen}}(x)$ is the density of $x = G(z)$

Some Issues of GAN

- The optimization procedure is unstable
  - one proposed improvement is WGAN (changing loss function)
- Practical implementation with logD trick is only a heuristic
  - inconsistent with the minimax formulation, implying the theory is flawed
- Minimizes JS divergence, not KL divergence

We want to design a procedure that addresses the above points.

- based on KL divergence minimization
- stable (by modifying optimization process of GAN)
- can explain the logD trick

New Approach

- Learn $G(z)$ to minimize the KL-divergence between the distributions of real data and generated data:

$$\int p_{\text{real}}(x) \log \frac{p_{\text{real}}(x)}{p_{\text{gen}}(x)} dx$$

- Procedure uses functional gradient learning greedily, similar to gradient boosting.
- Learning procedure uses functional compositions in the form

$$G_t(z) = G_{t-1}(z) + \eta_t g_t(G_{t-1}(z)), \qquad (t = 1, \ldots, T)$$

gradient descent in function space

Adversarial Learner

Given training examples $\{(x_i, y_i)\}$ (where $y_i = \pm 1$).

Assume there is an oracle adversarial learner $\mathcal{A}$ that can solve the logistic regression problem:

$$D(x) \approx \arg\min_D \sum_i \ln(1 + \exp(-D(x_i)y_i)).$$

Using

- real sample $S = \{x_1, \ldots, x_n\}$ with label $y = 1$
- generated sample $\{G(z_1), \ldots, G(z_m)\}$ with label $y = -1$

We can find approximately

$$D(x) \approx \ln \frac{p_*(x)}{p_{\text{gen}}(x)}$$

Johnson-Zhang'2018

Theorem

Consider variable transformation $X' = X + \eta g(X)$.
Let p *be the probability density of random variable* X.
Let p' *be the probability density of random variable* X'.
Let p_* *be the probability density of the real data.*
Let $\mathcal{D}(x) := \ln \frac{p_*(x)}{p(x)}$.
Assume $D_\epsilon(x) \approx \mathcal{D}(x)$ *is learned from adversarial learner:*

$$\int p_*(x) \max(1, \|\nabla \ln p_*(x)\|) \left(|D_\epsilon(x) - \mathcal{D}(x)| + \left| e^{D_\epsilon(x)} - e^{\mathcal{D}(x)} \right| \right) dx \leq \epsilon .$$

Then for some constant $c > 0$*:*

$$KL(p_*||p') \leq KL(p_*||p) - \eta \int p_*(x)\, g(x)^\top \nabla D_\epsilon(x)\, dx + c\eta^2 + c\eta\epsilon.$$

Implication of the Theorem

The result is

$$KL(p_*||p') \leq KL(p_*||p) - \eta \int p_*(x) g(x)^\top \nabla D(x)\, dx + O(\eta^2).$$

If we further take

$$g(x) = s(x)\nabla D(x),$$

where $s(x) > 0$ is a scaling factor, then

$$KL(p_*||p') \leq KL(p_*||p) - \eta \int p_*(x) s(x)\, \|\nabla D(x)\|_2^2\, dx + O(\eta^2).$$

Implication:

$$\int p_*(x) s(x)\, \|\nabla D(x)\|_2^2\, dx \to 0,$$

which means

$$D(x) = \ln \frac{p_*(x)}{p_{\text{gen}}(x)} = \text{constant}$$

Johnson-Zhang'2018

Algorithm 1 CFG: Composite Functional Gradient Learning

Require: real data $x_1^*, \ldots, x_n^*$, initial generator $G_0(z)$

1: **for** $t = 1, 2, \ldots, T$ **do**

2: $D_t(x) \leftarrow \arg\min_D \left[\frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-D(x_i^*)}) + \frac{1}{m}\sum_{i=1}^{m} \ln(1 + e^{D(G_{t-1}(z_i))})\right]$

3: $g_t(x) \leftarrow s_t(x)\nabla D_t(x)$ (usually $s_t(x) = 1$)

4: $G_t(z) \leftarrow G_{t-1}(z) + \eta_t g_t(G_{t-1}(z))$, for some $\eta_t > 0$.

5: **end for**

6: **return** generator $G_T(z)$

Theory: as $n \to \infty$ and $T \to \infty$

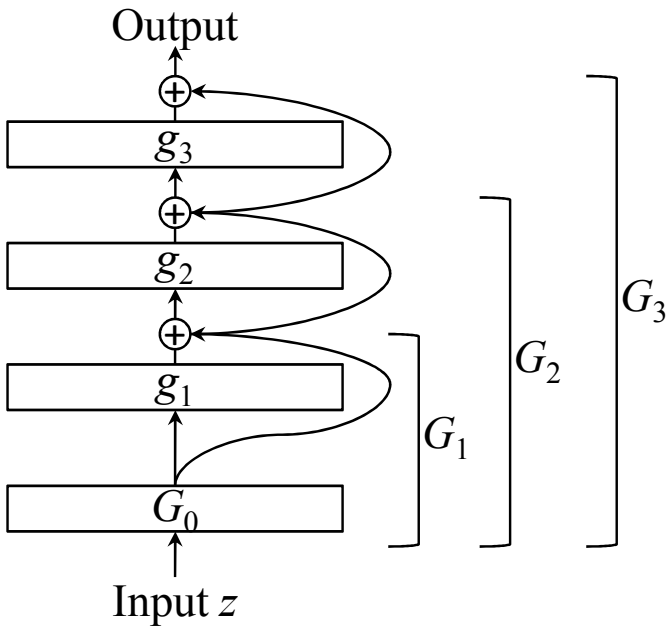
$$G_T(z) \sim p_*(\cdot)$$

Algorithm 2 ICFG: Incremental Composite Functional Gradient Learning

Require: training examples S_*, prior p_z, initial generator G_0, discriminator D

for $t = 1, 2, \ldots, T$ **do**
  for U steps **do**
    Sample $x_1^*, \ldots, x_b^*$ from S_*, and $z_1, \ldots, z_b$ according to p_z.
    Update discriminator D by SGD with minibatch gradient:
    $\nabla_{\theta_D} \frac{1}{b} \sum_{i=1}^{b} \left[\ln(1 + \exp(-D(x_i^*))) + \ln(1 + \exp(D(G_{t-1}(z_i))))\right]$
  end for
  $g_t(x) \leftarrow s_t(x)\nabla D(x)$ (most simply $s_t(x) = 1$)
  $G_t(z) \leftarrow G_{t-1}(z) + \eta_t g_t(G_{t-1}(z))$, for some $\eta_t > 0$.
end for
return generator G_T

Graphical Illustration



Generator network automatically derived by CFG
$T = 3$ and '$\oplus$' indicates addition

Problem: network depth grows when T increases
Solution: use a fixed depth approximator

Algorithm 3 xICFG: Approximate Incremental CFG Learning

Require: a set of training examples S_*, prior p_z, approximator $\widetilde{G}$ at its initial state, discriminator D.

1: **loop**
2: $S_z \leftarrow$ an input pool of the given size, sampled according to p_z.
3: $q_z \leftarrow$ the uniform distribution over S_z.
4: $G, D \leftarrow$ output of ICFG using S_*, q_z, $\widetilde{G}$, D as input.
5: **if** some exit criteria is met **then**
6: **return** generator G
7: **end if**
8: Update the approximator $\widetilde{G}$ to minimize $\frac{1}{2}\sum_{z \in S_z} \|\widetilde{G}(z) - G(z)\|^2$
9: **end loop**

$\widetilde{G}(z)$ is a network of the fixed size

$G(z)$ is of growing but limited size (initialized from $\widetilde{G}(z)$ every time)

Approximator Network

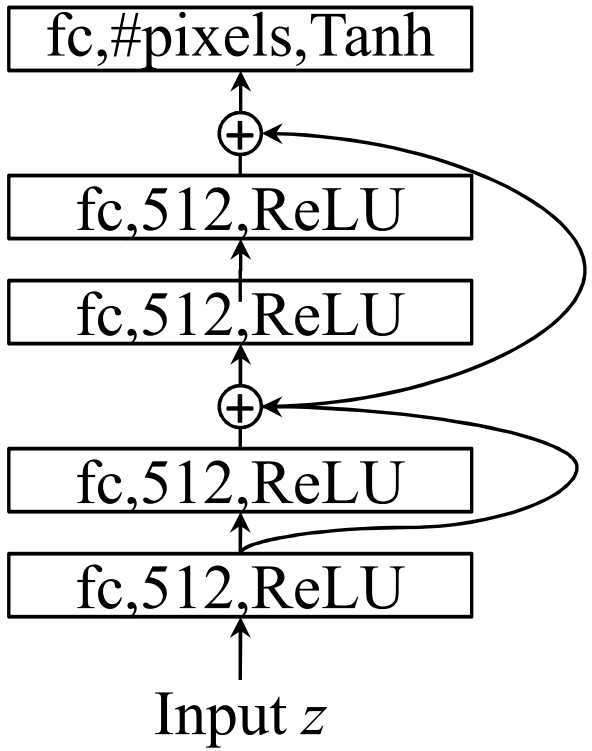


Figure: Fully-connected networks with shortcuts for use as a xICFG approximator or a GAN generator.

Comparison to Original GAN (GAN0)

Algorithm 4 GAN0(Generative Adversarial Nets)

Require: training examples S_*, prior p_z, discriminator D, generator G.

1: Initialize discriminator d and generator G randomly.
2: **for** T steps **do**
3: Update discriminator D by ascending the stochastic gradient:
4: Sample $z_1, \ldots, z_b$ according to p_z.
5: Update the generator by descending the stochastic gradient: $\frac{1}{b} \sum_{i=1}^{b} \underbrace{(1 + \exp(-D(G(z_i))))^{-1}}_{s(G(z_i))} \nabla_{\theta_G} D(G(z_i))$
6: **end for**
7: **return** generator G

Problem: $s(G(z)) \approx 0$ when $D(G(z)) \ll 0$, which happens in the beginning.

Comparison to GAN with LogD trick (GAN1)

Algorithm 5 GAN1(Generative Adversarial Nets)

Require: training examples S_*, prior p_z, discriminator D, generator G.

1: Initialize discriminator d and generator G randomly.
2: **for** T steps **do**
3: Update discriminator d by ascending the stochastic gradient:
4: Sample $z_1, \ldots, z_b$ according to p_z.
5: Update the generator by descending the stochastic gradient: $\frac{1}{b}\sum_{i=1}^{b} \underbrace{(1+\exp(D(G(z_i))))^{-1}}_{s(S(z_i))} \nabla_{\theta_G} D(G(z_i))$
6: **end for**
7: **return** generator G

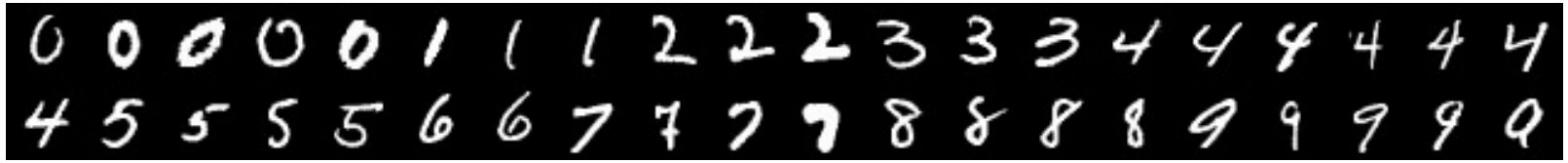
Good! $s(G(z)) \approx 1$ when $D(G(z)) \ll 0$, which happens in the beginning.

Experiments

- GAN0 is original GAN
  - SGD optimization of the minimax formulation of GAN
- GAN1 is GAN with log trick, which is a heuristics
  - cannot be explained using the original minimax formulation of GAN can be explained by our theory
- xICFG
  - similar to GAN1
  - grows generator using composite function gradient
  - use approximator to reduce the network to fixed size

Data

- MNIST



- Street View House Numbers

- large-scale scene understanding (LSUN)

- quality *inception score*:

$$\exp(\mathbb{E}_x \mathrm{KL}(\Pr(y|x)||\Pr(y)))$$

  - high-quality images should lead to high confidence in classification, and high inception score.
- diversity *diversity score*:

$$\exp(-\mathrm{KL}(\Pr(y)||\Pr_*(y))$$

  - diversity of generated images measured by diversity score becomes large (approaching to 1) when generated class distribution mimics real class distribution.

LSUN Performance Comparisons

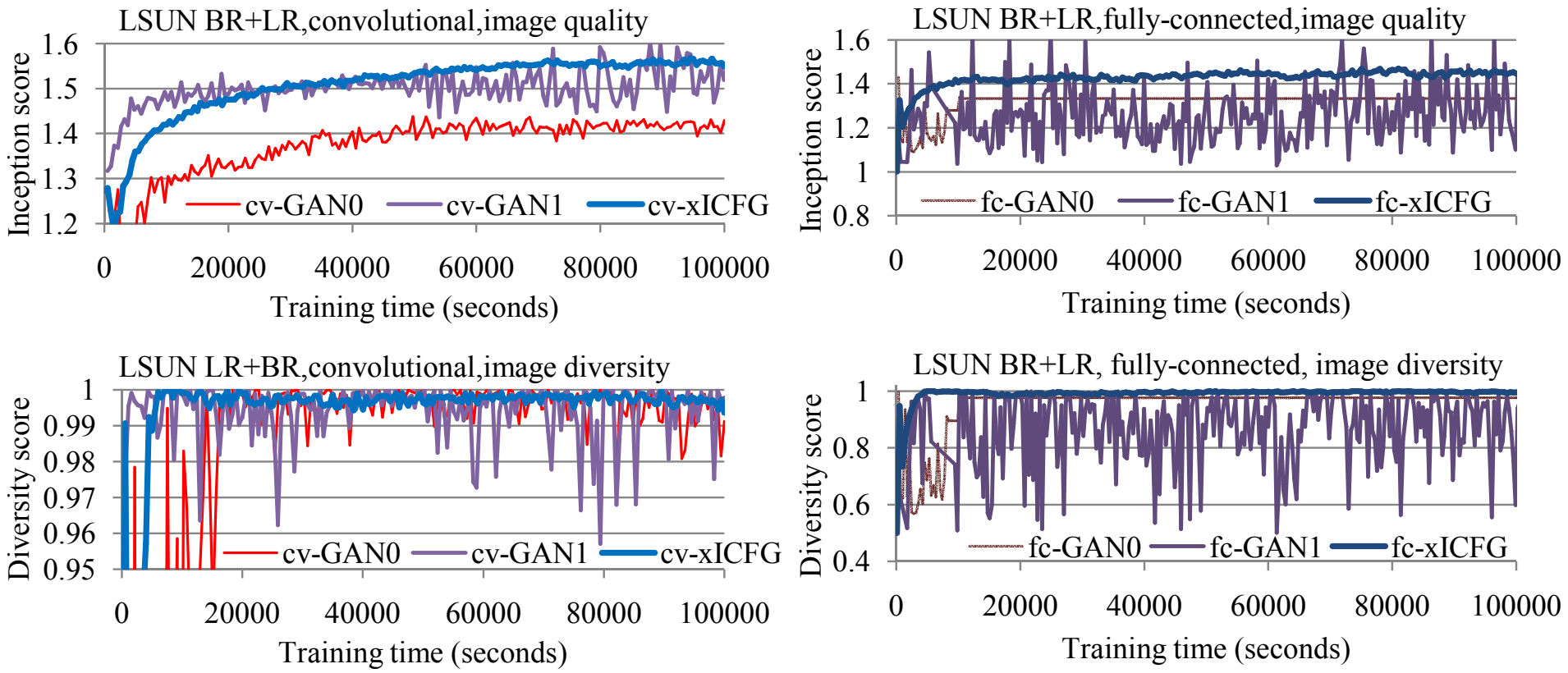


Figure: LSUN BR+LR. Image quality (upper) and diversity (lower). With the convolutional (left) and the fully-connected (right) approximator/generator.

Generated Data (LSUN)

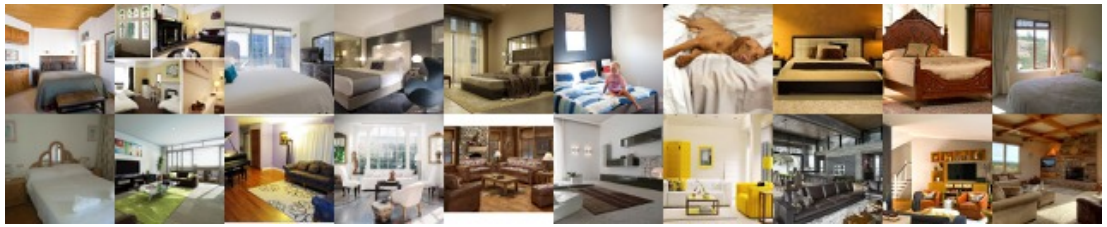


Figure: LSUN bedrooms & living rooms. Real images from the training set.

(a) cv-xICFG (1.55). High quality. No signs of modal collapse.

Wasserstein GAN (WGAN)?

- WGAN uses a different loss function than log-loss
- It claims to improve stability of GAN
- The CFG procedure claims improved stability as well
  - by making optimization easier
- How does CFG compare to WGAN?

Comparison with WGAN: CNN

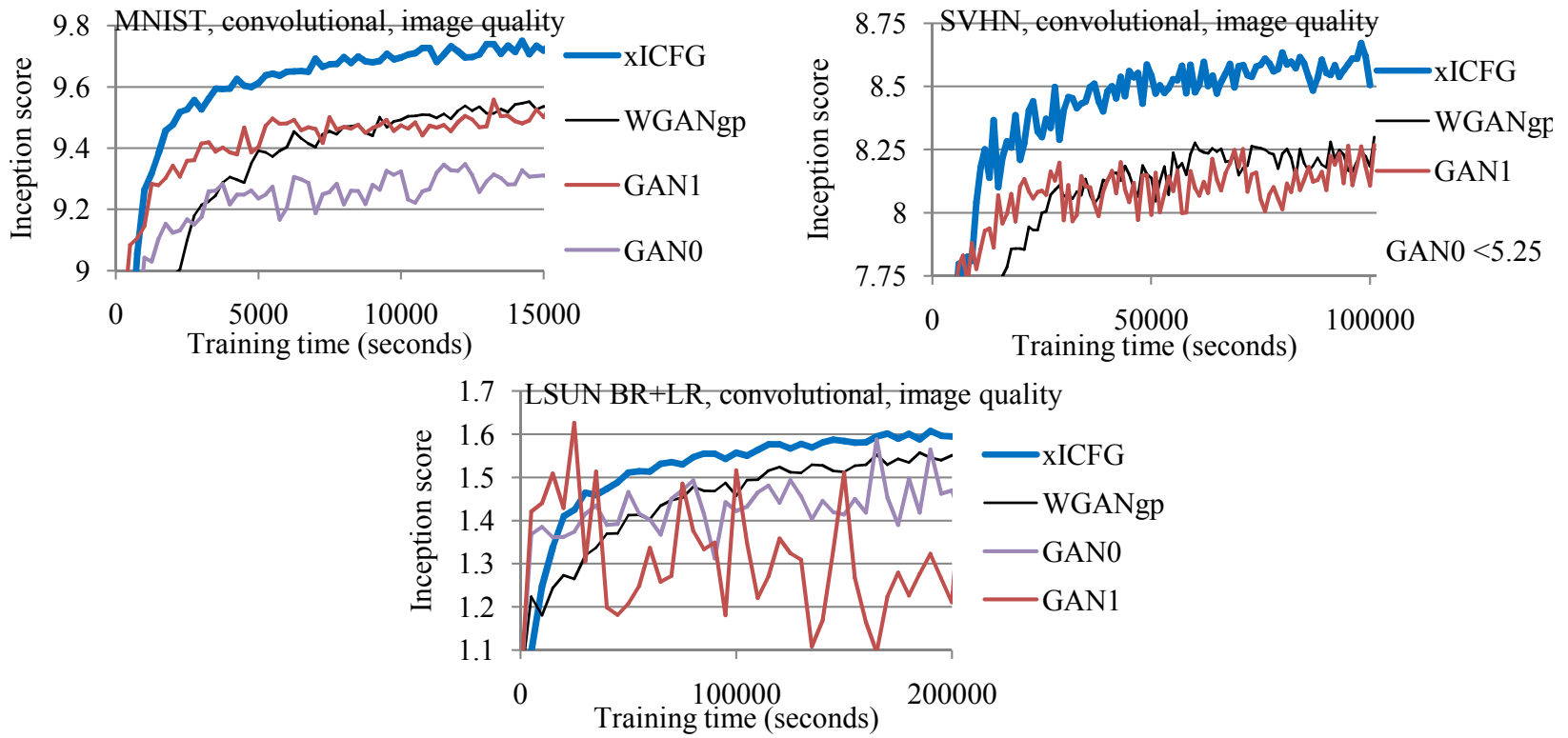


Figure: Image quality. Convolutional networks. The legends are sorted from the best to the worst. xICFG outperforms the others.

Creativity in LSUN (Tower & Bridge)

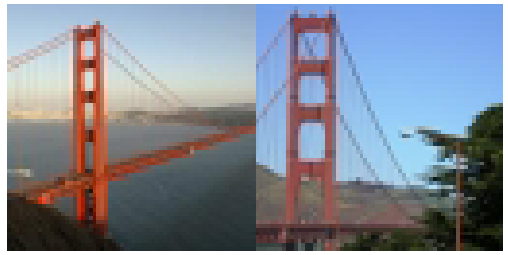
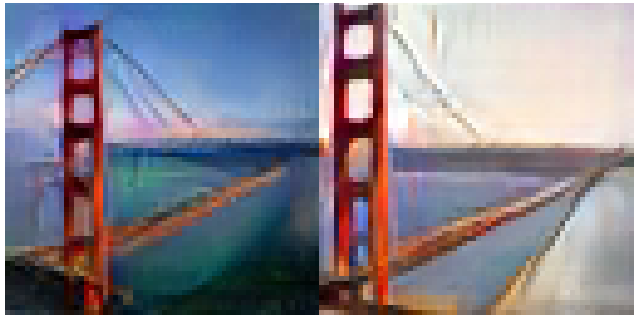
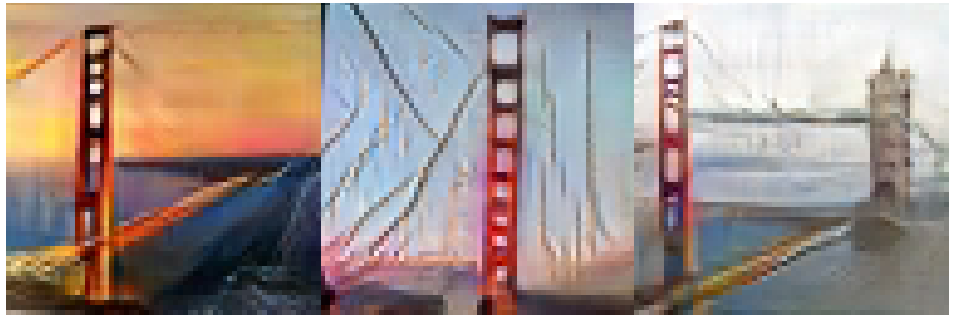


Figure: Real Golden Gate Bridge images: the red tower has 4 grids



(a) "Realistic"



(b) "Creative"

(a) Images generated by xICFG that resemble Golden Gate Bridge

(b) Images generated by xICFG that *modify* Golden Gate Bridge

Summary

- In the generative adversarial learning setting:
  GAN's minimax formulation optimizes JS-divergence
  GAN's algorithm is not stable.
  the practical use of logD trick inconsistent with the minimax formulation
- The optimization problem is hard and unstable
- This work: change optimization, gradient descent in function space
- Learn generator $G(z)$ using CFG (composite functional gradient).
  - Theory: minimizes KL-divergence
  - Theory: lead to the new stable algorithm xICFG.
  - Theory: explains the logD trick of GAN.
  - Experiments: xICFG performs better than GAN/WGAN
- Reference: *Rie Johnson, Tong Zhang*, Composite Functional Gradient Learning of Generative Adversarial Models. [arXiv:1801.06309]

Thank you!