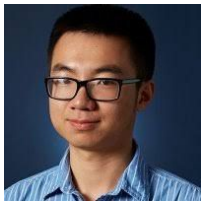
CS294-158 Deep Unsupervised Learning

Lecture 6: Implicit Models (Continued)

Pieter Abbeel, Peter Chen, Jonathan Ho, Aravind Srinivas

UC Berkeley

So far...

- Autoregressive models
  - MADE, PixelRNN/CNN, Gated PixelCNN, PixelSNAIL, SPN
- Flow models
  - NICE, Autoregressive Flows, RealNVP, Glow, Flow++
- Latent Variable Models
  - Wake Sleep, VAE, IWAE, IAF-VAE

So far...

- Autoregressive models
  - MADE, PixelRNN/CNN, Gated PixelCNN, PixelSNAIL, SPN
- Flow models
  - NICE, Autoregressive Flows, RealNVP, Glow, Flow++
- Latent Variable Models
  - Wake Sleep, VAE, IWAE, IAF-VAE

- Common aspect: Likelihood-based models - exact (autoregressive and flows) or approximate (VAE / IAF-VAE).

Building a sampler

- All these models (PixelCNN, VAE, RealNVP) are generative models apart from being useful for compression.
- What does generative mean?
  - You can sample from the model and the distribution of samples approximates the distribution of true data points.

Building a sampler

- All these models (PixelCNN, VAE, RealNVP) are generative models apart from being useful for compression.
- What does generative mean?
  - You can sample from the model and the distribution of samples approximates the distribution of true data points.
- Question -
  - For building a generative model (leaving aside utility for compression), i.e, a sampler which spits out samples that look like true data points, do we need to optimize log-likelihood?

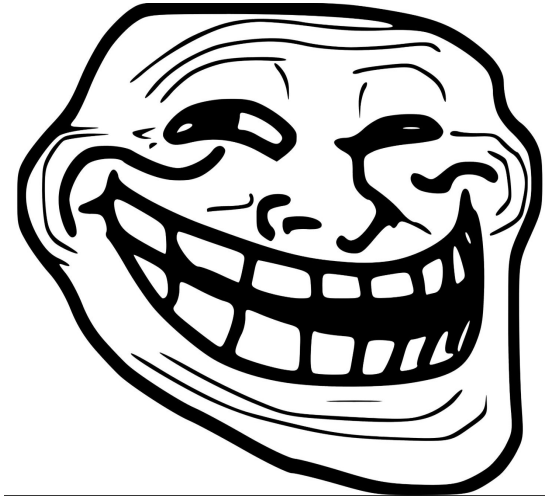
Building a sampler

- How about this sampler?

```
import glob, cv2
files = glob.glob('*.jpg')
def _sample():
    idx = np.random.randint(len(files))
    return cv2.imread(files[idx])
def sample(*, n_samples):
    samples = np.array([_sample() for _ in range(n_samples)])
    return samples
```

Building a sampler

- How about this sampler?



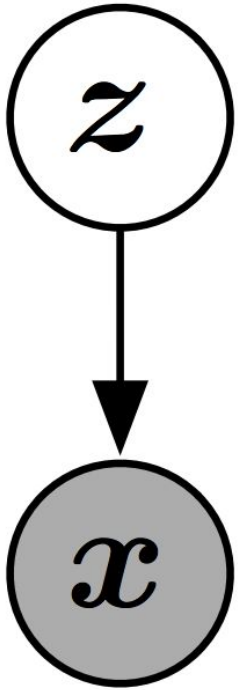
```
import glob, cv2
files = glob.glob('*.jpg')
def _sample():
    idx = np.random.randint(len(files))
    return cv2.imread(files[idx])
def sample(*, n_samples):
    samples = np.array([_sample() for _ in range(n_samples)])
    return samples
```

Building a sampler

- You don’t just want to sample the exact data points you have.
- You want to build a generative model that can understand the underlying distribution of data points and
  - smoothly interpolate across the training samples
  - output samples similar but not the same as training data samples
  - outputs samples representative of the underlying factors of variation in the training distribution.
  - Example: digits with unseen strokes, faces with unseen poses, etc.

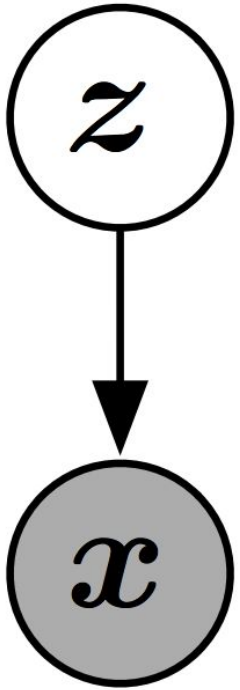
Implicit Models

- Sample z from a fixed noise source distribution (uniform or gaussian).
- Pass the noise through a deep neural network to obtain a sample x.



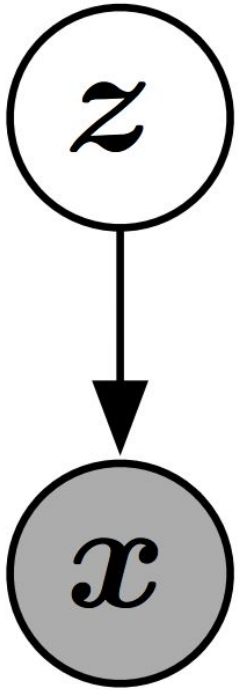
Implicit Models

- Sample z from a fixed noise source distribution (uniform or gaussian).
- Pass the noise through a deep neural network to obtain a sample x.
- Sounds too familiar?



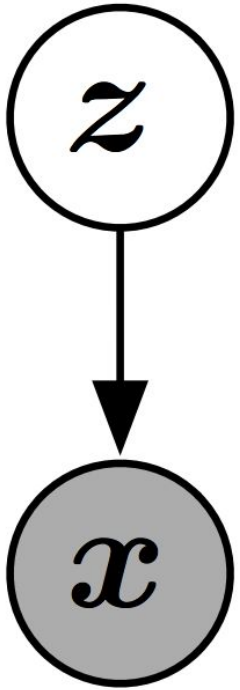
Implicit Models

- Sample z from a fixed noise source distribution (uniform or gaussian).
- Pass the noise through a deep neural network to obtain a sample x.
- Sounds too familiar?
  - We saw this Bayes Net in
    - Flows Models
    - VAE
- What's going to be different here?



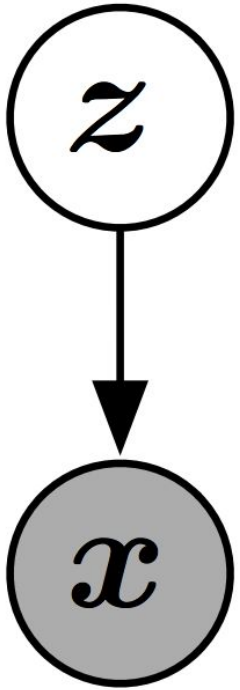
Implicit Models

- Sample z from a fixed noise source distribution (uniform or gaussian).
- Pass the noise through a deep neural network to obtain a sample x.
- Sounds too familiar?
  - We saw this Bayes Net in
    - Flows Models
    - VAE
- What's going to be different here?
  - Learning the deep neural network **without** *explicit* density estimation.



Implicit Models

- Sample z from a fixed noise source distribution (uniform or gaussian).
- Pass the noise through a deep neural network to obtain a sample x.
- Learn the deep neural network **without** *explicit* density estimation.
- How do we design an objective function?



Implicit Models

- Given samples from data distribution $p_{\text{data}} : x_1, x_2, \cdots, x_n$
- Given a sampler $q_\phi(z) = \text{DNN}(z; \phi)$ where $z \sim p(z)$
- $x = q_\phi(z)$ induces a density function p_{model}
- Do not have an explicit form for p_{data} or p_{model}; can only draw samples
- Make p_{model} as close to p_{data} as possible by learning an appropriate ϕ

Departure from maximum likelihood

- We need some measure of how far apart p_{data} and induced p_{model} are
- With density models, we used $KL(p_{\text{data}}||p_{\text{model}})$ which gave us the objective $\mathbb{E}_{x\sim p_{\text{data}}}[\log p_\theta(x)]$ (discarding the term independent of θ) where we explicitly modeled p_{model} as $p_\theta(x)$
- Not having an explicit $p_\theta(x)$ requires us to come up distance measures that potentially behave differently from maximum likelihood.
- Example, Maximum Mean Discrepancy (MMD); Jensen Shannon Divergence (JSD); Earth Mover's Distance, etc.

Moment Matching

- Key idea: Match the moments of the data and model distributions to bring them closer
- Called the two-sample test in hypothesis testing

Given samples $X = \{x_i\}_{i=1}^{N}$ from P_X , $Y = \{y_j\}_{j=1}^{M}$ from P_Y , how do we compare P_X and P_Y?

- Not feasible to compute higher order moments in high dimensions

Moment Matching

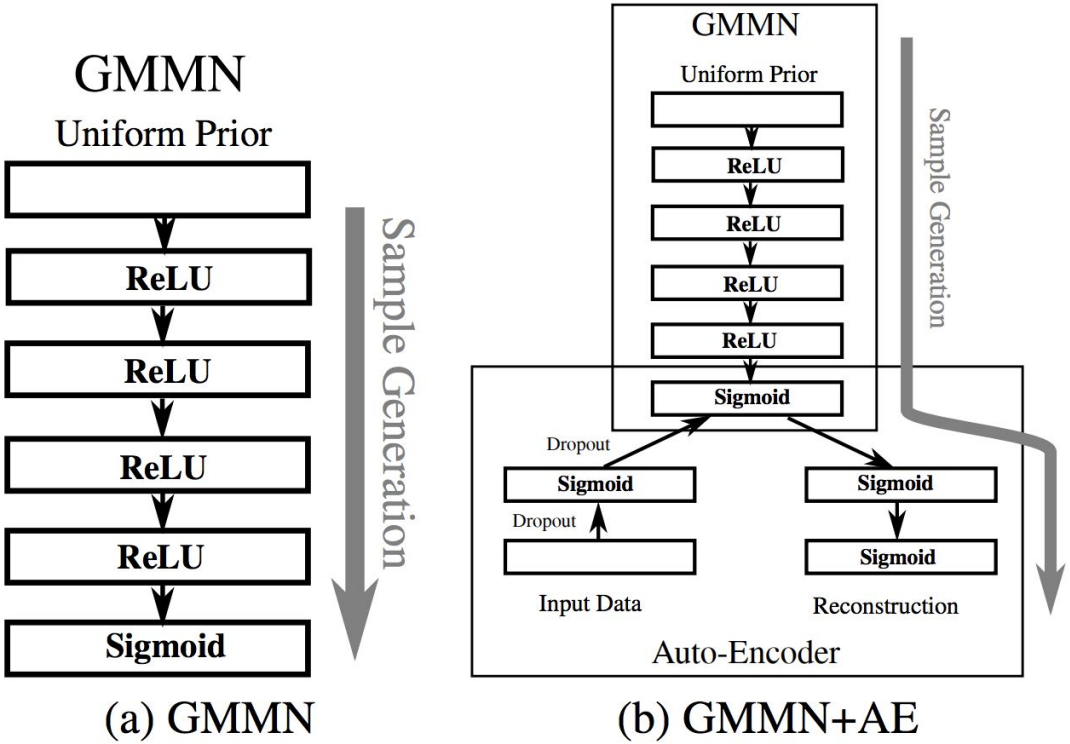
- Kernel trick

$$
\begin{aligned}
L_{MMD} &= \| \frac{1}{N} \sum_{i=1}^{N} \phi(x_i) - \frac{1}{M} \sum_{j=1}^{M} \phi(y_j) \|^2 \\
&= \frac{1}{N^2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \phi(x_i)^T \phi(x_i') + \frac{1}{M^2} \sum_{j=1}^{M} \sum_{j'=1}^{M} \phi(y_j)^T \phi(y_j') - \frac{2}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} \phi(x_i)^T \phi(y_j) \\
&= \frac{1}{N^2} \sum_{i=1}^{N} \sum_{i'=1}^{N} k(x_i, x_{i'}) - \frac{2}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} k(x_i, y_j) + \frac{1}{M^2} \sum_{j=1}^{M} \sum_{j'=1}^{M} k(y_j, y_{j'})
\end{aligned}
$$

Moment Matching

- Identity function ϕ is equivalent to matching sample means
- Kernel trick allows you to optimize mean discrepancy in higher dimensional spaces.
- Example $k(x, x') = \exp\left(\frac{-||x - x'||^2}{2\sigma}\right)$
- Refer to Gretton et al 2007, 2012 - Mathematical Treatment

Generative Moment Matching Networks



Li, Swersky, Zemel (2015)

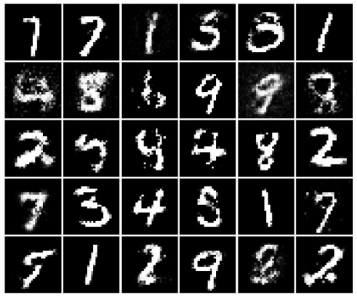
Generative Moment Matching Networks

Input : Dataset $\{\mathbf{x}_1^d, ..., \mathbf{x}_N^d\}$, prior $p(\mathbf{h})$, network $f(\mathbf{h}; \mathbf{w})$ with initial parameter $\mathbf{w}^{(0)}$

Output: Learned parameter $\mathbf{w}^*$

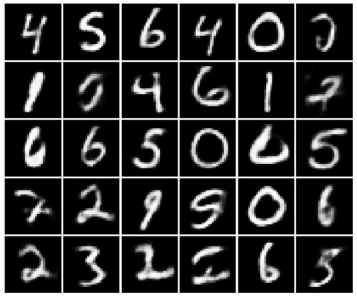
while *Stopping criterion not met* **do**
  Get a minibatch of data $\mathbf{X}^d \leftarrow \{\mathbf{x}_{i_1}^d, ..., \mathbf{x}_{i_b}^d\}$
  Get a new set of samples $\mathbf{X}^s \leftarrow \{\mathbf{x}_1^s, ..., \mathbf{x}_b^s\}$
  Compute gradient $\frac{\partial \mathcal{L}_{\text{MMD}}}{\partial \mathbf{w}}$ on $\mathbf{X}^d$ and $\mathbf{X}^s$
  Take a gradient step to update $\mathbf{w}$
end

Li, Swersky, Zemel (2015)



(a) GMMN MNIST samples

(b) GMMN TFD samples



(c) GMMN+AE MNIST samples

(d) GMMN+AE TFD samples

Generative Moment Matching Networks

- Need a good kernel for the mean discrepancy measure

> For universal kernels like the Gaussian kernel, defined as $k(x, x') = \exp(-\frac{1}{2\sigma}|x - x'|^2)$, where σ is the bandwidth parameter, we can use a Taylor expansion to get an explicit feature map ϕ that contains an infinite number of terms and covers all orders of statistics. Minimizing MMD under this feature expansion is then equivalent to minimizing a distance between *all* moments of the two distributions.

- Not shown to scale well beyond MNIST and TFD (and some variants on CIFAR 10 later) - needs autoencoding, large minibatch and mixture of kernels with different bandwidths

Generative Adversarial Networks

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. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

Generative Adversarial Networks

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

- Two player minimax game between generator (G) and discriminator (D)
- (D) tries to maximize the log-likelihood for the binary classification problem [data: real (1), generated: fake (0)]
- (G) tries to minimize the log-probability of its samples being classified as “fake” by the discriminator (D)

Generative Adversarial Networks

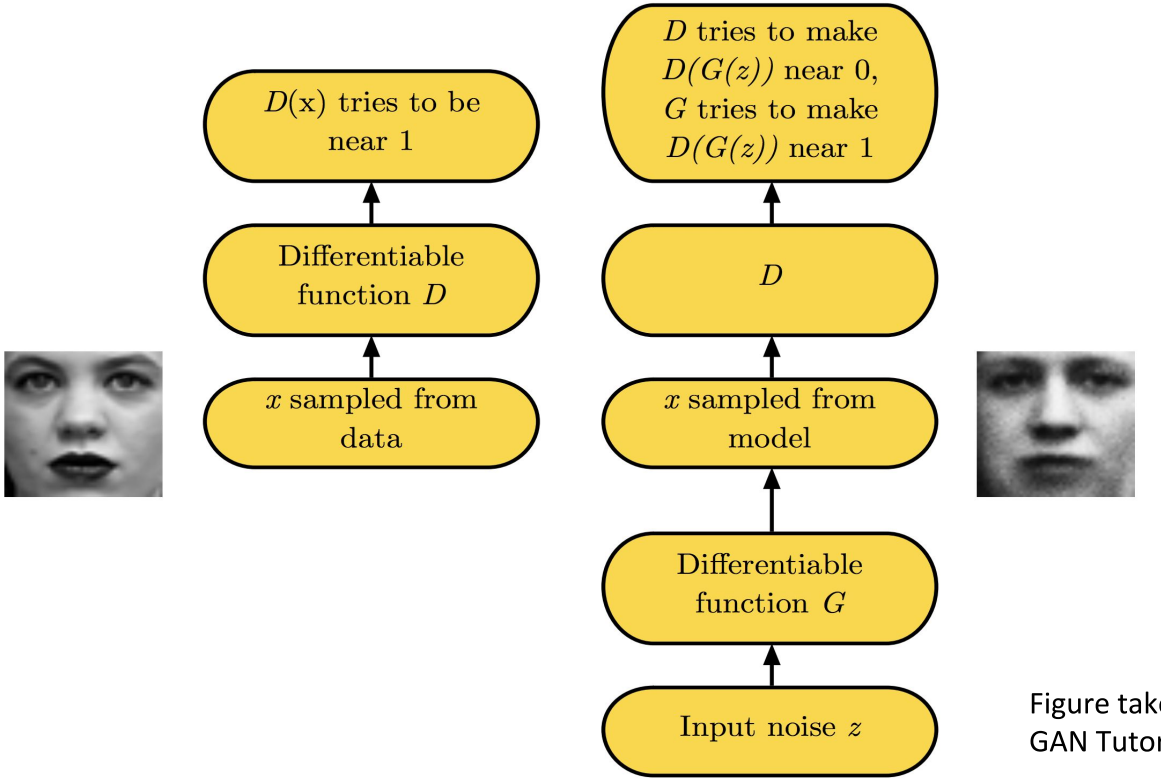


Figure taken from NeurIPS 2016 GAN Tutorial (Goodfellow)

GAN

See it in action

Generative Adversarial Networks

- What's the optimal discriminator given generated and true distributions?

$$
\begin{aligned}
V(G,D) &= \mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log(1-D(G(z)))\right] \\
&= \int_x p_{\text{data}}(x)\log D(x)dx + \int_z p(z)\log(1-D(G(z)))dz \\
&= \int_x p_{\text{data}}(x)\log D(x)dx + \int_x p_g(x)\log(1-D(x))dx \\
&= \int_x \left[p_{\text{data}}(x)\log D(x) + p_g(x)\log(1-D(x))\right]dx
\end{aligned}
$$

$$
\nabla_y \left[a\log y + b\log(1-y)\right] = 0 \implies y^* = \frac{a}{a+b} \quad \forall \quad [a,b]\in\mathbb{R}^2\backslash[0,0]
$$

$$
\implies D^*(x) = \frac{p_{\text{data}}(x)}{(p_{\text{data}}(x)+p_g(x))}
$$

Generative Adversarial Networks

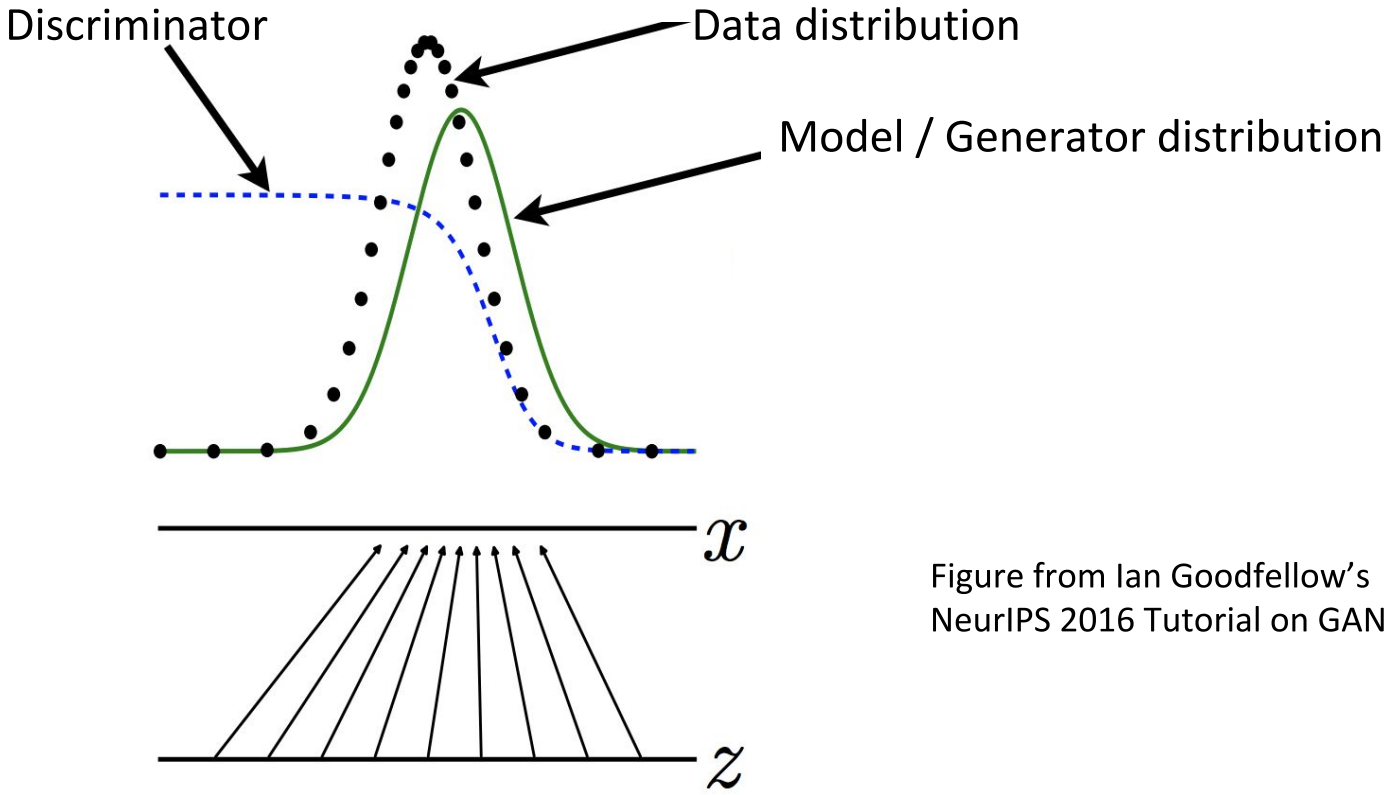


Figure from Ian Goodfellow's NeurIPS 2016 Tutorial on GANs

Generative Adversarial Networks

- What is the generator objective given the optimal discriminator?

$$
\begin{aligned}
V(G, D^*) &= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log D^*(x) \right] + \mathbb{E}_{x \sim p_g} \left[\log (1 - D^*(x)) \right] \\
&= \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g} \left[\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} \right] \\
&= -\log(4) + \underbrace{KL\left(p_{\text{data}} \| \left(\frac{p_{\text{data}} + p_g}{2} \right) \right) + KL\left(p_{\text{g}} \| \left(\frac{p_{\text{data}} + p_g}{2} \right) \right)}_{\text{(Jensen-Shannon Divergence (JSD) of } p_{\text{data}} \text{ and } p_g) \geq 0}
\end{aligned}
$$

$$
V(G^*, D^*) = -\log(4) \text{ when } p_g = p_{\text{data}}
$$

Behaviors across divergence measures

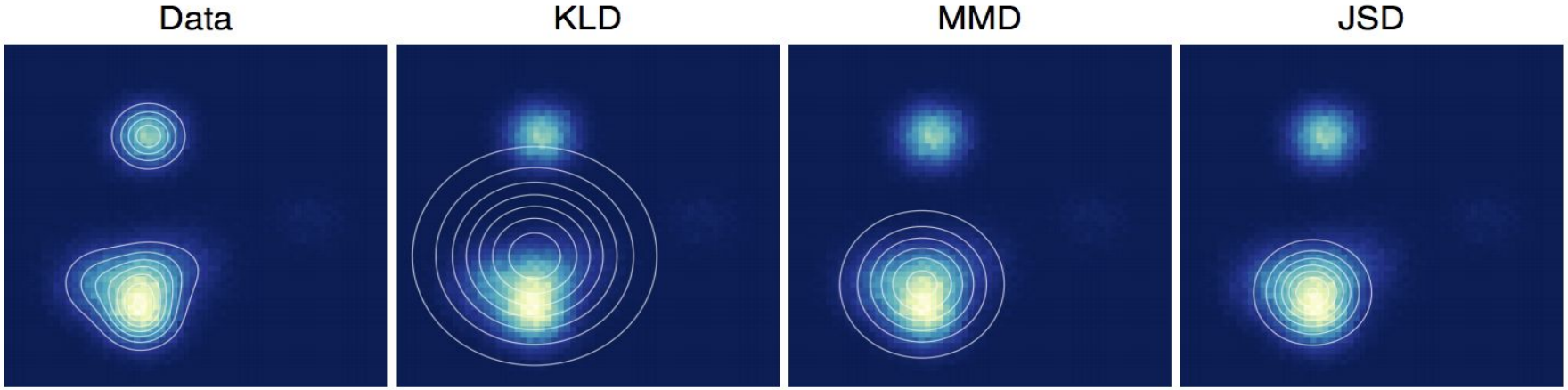
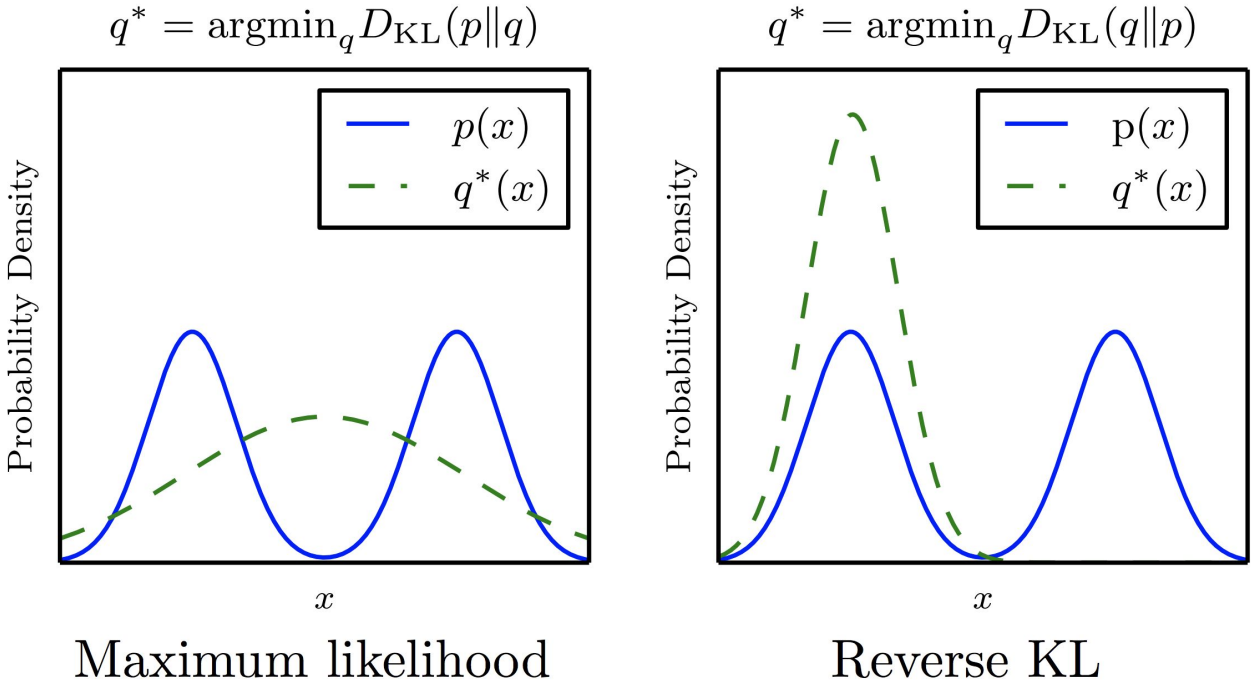


Figure 1: An isotropic Gaussian distribution was fit to data drawn from a mixture of Gaussians by either minimizing Kullback-Leibler divergence (KLD), maximum mean discrepancy (MMD), or Jensen-Shannon divergence (JSD). The different fits demonstrate different tradeoffs made by the three measures of distance between distributions.

A note on the evaluation of generative models (Theis, Van den Oord, Bethge 2015)

Direction of KL divergence



Deep Learning Textbook (Goodfellow 2016)- Chapter 3

Mode covering vs Mode seeking: Tradeoffs

- For compression, one would prefer to ensure all points in the data distribution are assigned probability mass.
- For generating good samples, blurring across modes spoils perceptual quality because regions outside the data manifold are assigned non-zero probability mass.
- Picking one mode without assigning probability mass on points outside can produce “better-looking” samples.
- Caveat: More expressive density models can place probability mass more accurately. Example: Using mixture of gaussians as opposed to a single isotropic gaussian.

Back to GANs

Recall

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

Mini-Exercise:

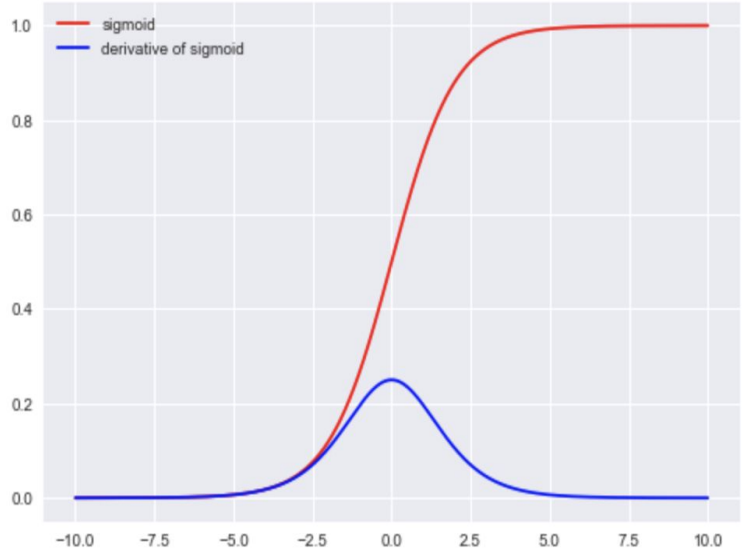
- Is it feasible to run the inner optimization to completion?
- For this specific objective, would it create problems if we were able to do so?

Back to GANs

- Generator samples confidently classified as fake by the discriminator receive no gradient for the generator update.
- Referred to as the 'Discriminator Saturation' problem.

$$\nabla_{G(z)} \log(1 - D(G(z))) \text{ where } D(x) = \text{sigmoid}(x; \theta) = \sigma(x; \theta)$$

$$\nabla_x \sigma(x) = \sigma(x)(1 - \sigma(x))$$



Back to GANs

- Alternate between optimizing (taking gradient descent steps on) the discriminator and generator objectives

$$L^{(D)}(\theta_D, \theta_G) = -\mathbb{E}_{x \sim p_{\text{data}}} \left[\log D(x; \theta_D)\right] - \mathbb{E}_{z \sim p(z)} \left[\log(1 - D(G(z; \theta_G), \theta_D))\right]$$

$$L^{(G)}(\theta_D, \theta_G) = \mathbb{E}_{z \sim p(z)} \left[\log(1 - D(G(z; \theta_G), \theta_D))\right]$$

$$\theta_D := \theta_D - \alpha^{(D)} \nabla_{\theta_D} L^{(D)}(\theta_D, \theta_G)$$
$$\theta_G := \theta_G - \beta^{(G)} \nabla_{\theta_G} L^{(G)}(\theta_D, \theta_G)$$

- Balancing these two updates is hard for the zero-sum game
- Goodfellow suggests modifying the generator objective to make the adversarial game non-zero sum and help address the saturation problem

GANs - Pseudocode

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**

  - 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

  - 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:

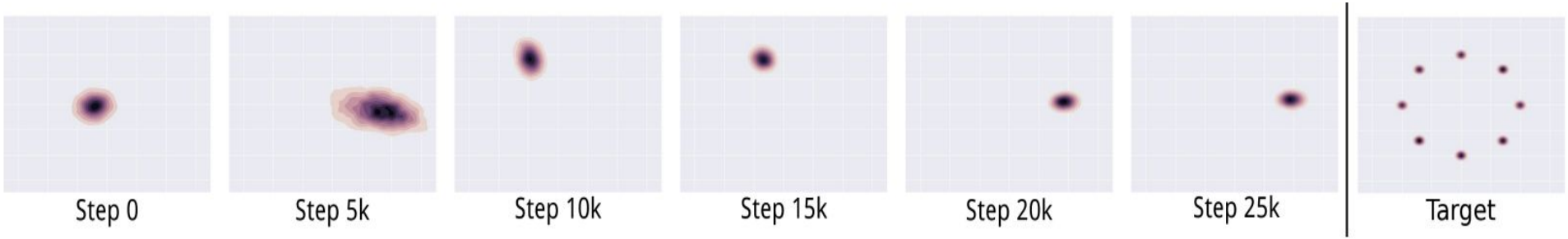
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

(Taken from Goodfellow et al 2014)

Mode Collapse



Standard GAN training collapses when the true distribution is a mixture of gaussians (Figure from Metz et al 2016)

GANs - Non Saturating version

$$L^{(D)} = -\mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] - \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$$

$$L^{(G)} = -L^{D} \equiv \min_{G} \mathbb{E}_{z \sim p(z)} \log(1 - D(G(z)))$$

Not zero-sum

$$L^{(D)} = -\mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] - \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$$

$$L^{(G)} = -\mathbb{E}_{z \sim p(z)} \log(D(G(z)) \equiv \max_{G} \mathbb{E}_{z \sim p(z)} \log(D(G(z))$$

GAN samples from 2014

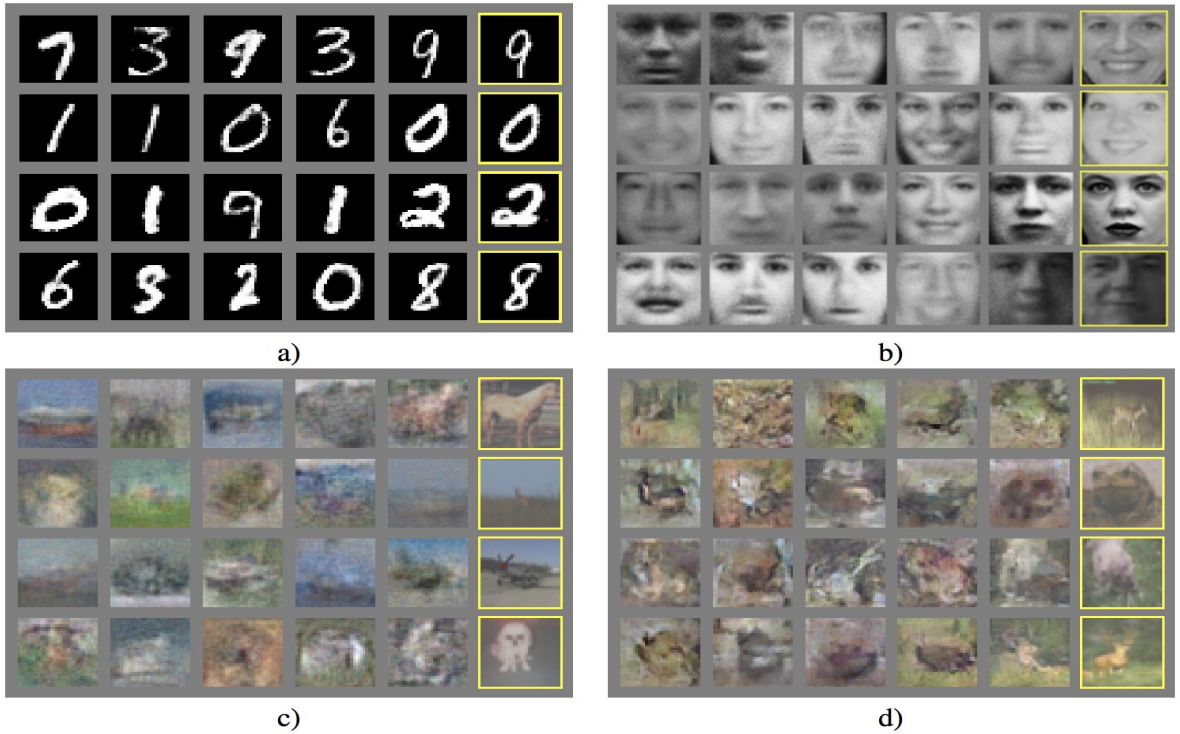


Figure from Goodfellow et al 2014

How to evaluate?

- Evaluation for GANs is still an open problem
- Unlike density models, you cannot report explicit likelihood estimates on test sets.

Parzen-Window density estimator

- Also known as Kernel Density Estimator (KDE)
- An estimator with kernel K and bandwidth h:

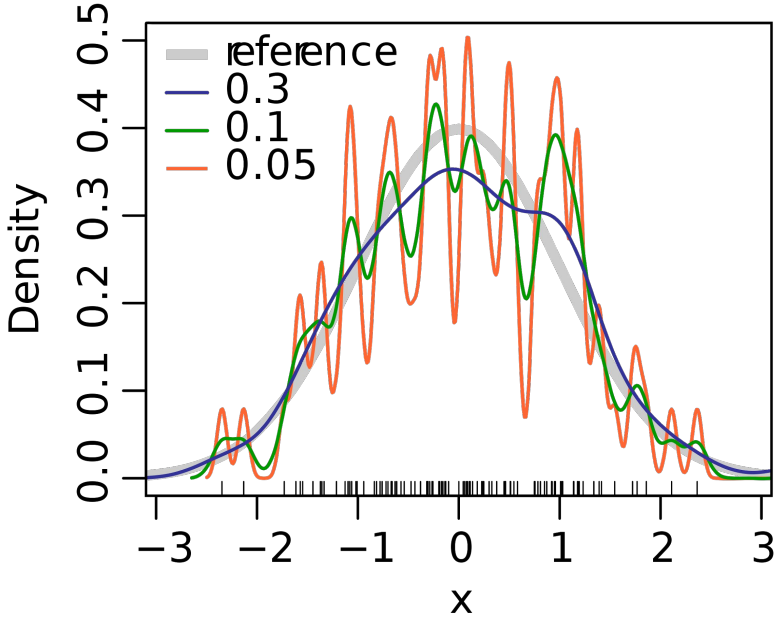
$$\hat{p_h}(x) = \frac{1}{nh} \sum_i K\left(\frac{x - x_i}{h}\right)$$

- In generative model evaluation, K is usually density function of standard Normal distribution

Bishop 2006

Parzen-Window density estimator

- Bandwidth h matters
- Bandwidth h chosen according to validation set



Bishop 2006

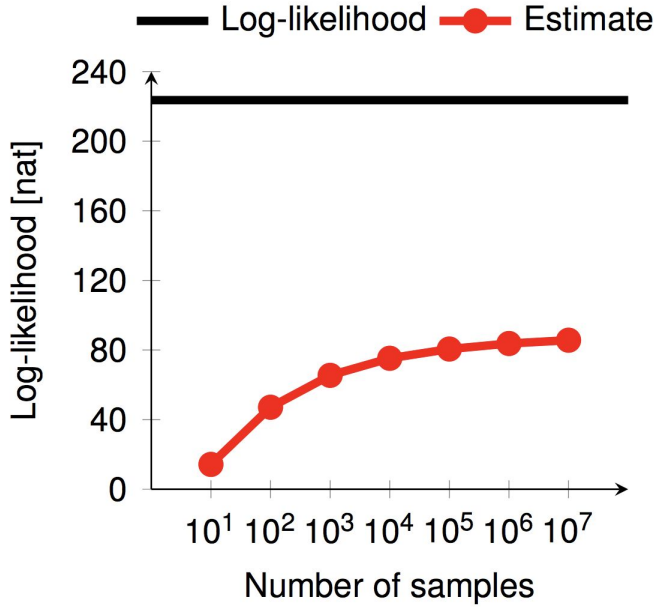
Evaluation

Model	MNIST	TFD
DBN [3]	138 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	$\mathbf{2110 \pm 50}$
Deep GSN [5]	214 ± 1.1	1890 ± 29
Adversarial nets	$\mathbf{225 \pm 2}$	$\mathbf{2057 \pm 26}$

Parzen Window density estimates (Goodfellow et al)

Parzen-Window density estimator

- Parzen Window estimator can be unreliable



Model	Parzen est. [nat]
Stacked CAE	121
DBN	138
GMMN	147
Deep GSN	214
Diffusion	220
GAN	225
True distribution	**243**
GMMN + AE	282
k-means	313

A note on the evaluation of generative models (Theis, Van den Oord, Bethge 2015)

Inception Score

- Can we side-step high-dim density estimation?
- One idea: good generators generate samples that are semantically diverse
- Semantics predictor: trained Inception Network v3
  - p(y|x), y is one of the 1000 ImageNet classes
- Considerations:
  - each image x should have distinctly recognizable object -> p(y|x) should have low entropy
  - there should be as many classes generated as possible -> p(y) should have high entropy

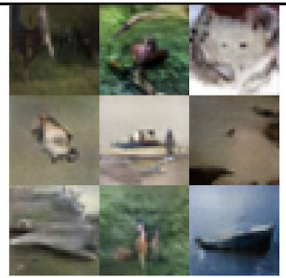
Inception Score

- Inception model: $p(y|x)$
- Marginal label distribution: $p(y) = \int_x p(y|x)p_g(x)$
- Inception Score:

$$\begin{aligned}\mathrm{IS}(x) =& \exp(\mathbb{E}_{x\sim p_g}\left[D_{\mathrm{KL}}\left[p(y|x) \parallel p(y)\right]\right]) \\ =& \exp(\mathbb{E}_{x\sim p_g, y\sim p(y|x)}\left[\log p(y|x) - \log p(y)\right]) \\ =& \exp(H(y) - H(y|x))\end{aligned}$$

Improved Gan (Salimans et al 2016)

Inception Score

Samples				
Model	Real data	Our methods	-VBN+BN	-L+HA
Score $\pm$ std.	$11.24 \pm .12$	$8.09 \pm .07$	$7.54 \pm .07$	$6.86 \pm .06$

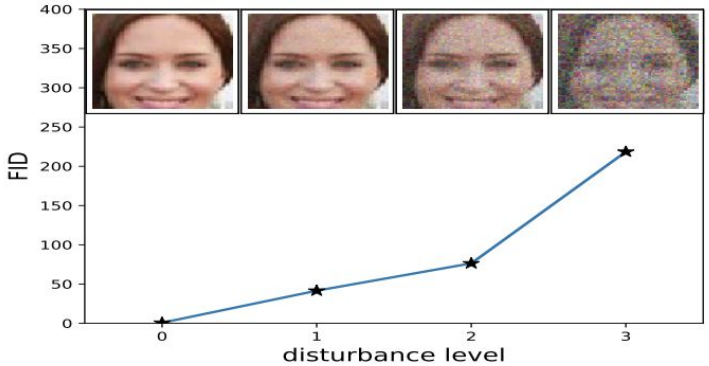
Fréchet Inception Distance

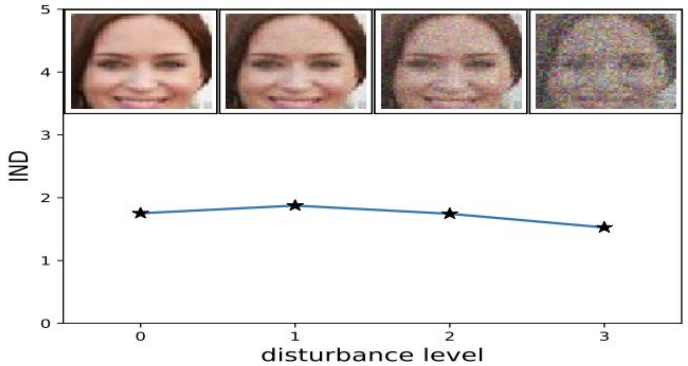
- Inception Score doesn't sufficiently measure diversity: a list of 1000 images (one of each class) can obtain perfect Inception Score
- FID was proposed to capture more nuances
- Embed image x into some feature space (2048-dimensional activations of the Inception-v3 pool3 layer), then compare mean (m) & covariance (C) of those random features

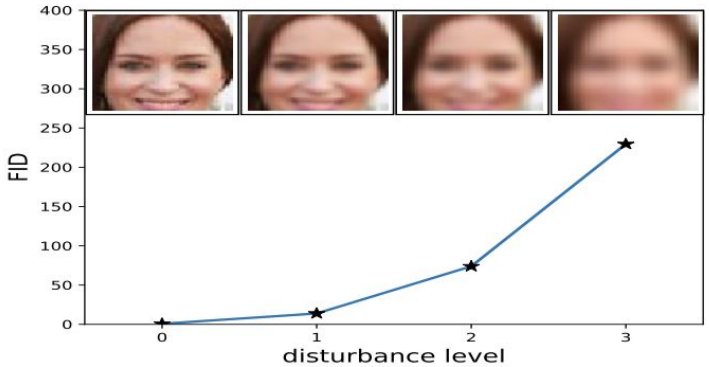
$$d^2((\boldsymbol{m}, \boldsymbol{C}), (\boldsymbol{m}_w, \boldsymbol{C}_w)) = \|\boldsymbol{m} - \boldsymbol{m}_w\|_2^2 + \mathrm{Tr}\big(\boldsymbol{C} + \boldsymbol{C}_w - 2\big(\boldsymbol{C}\boldsymbol{C}_w\big)^{1/2}\big)$$

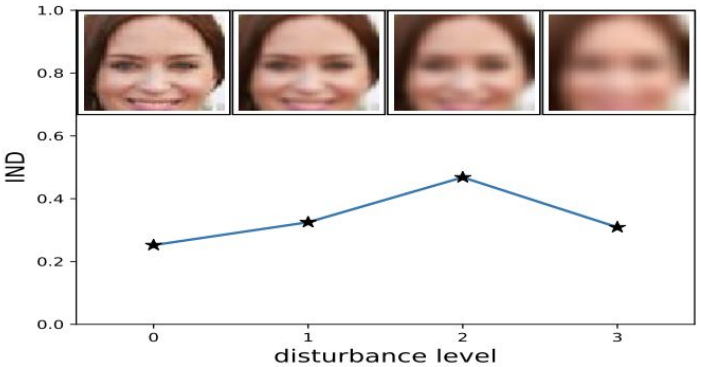
(Heusel et al, 2017)

Fréchet Inception Distance



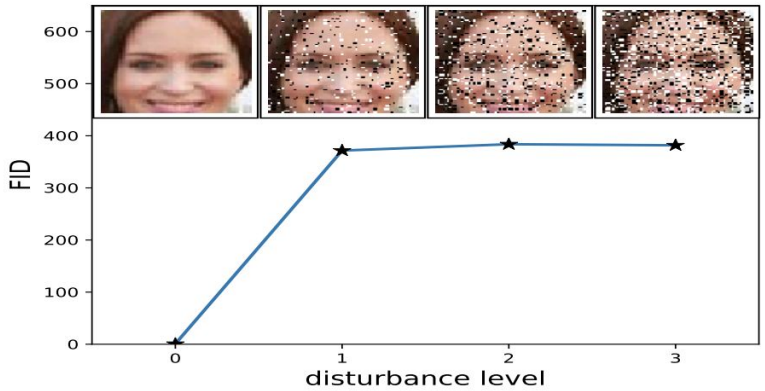


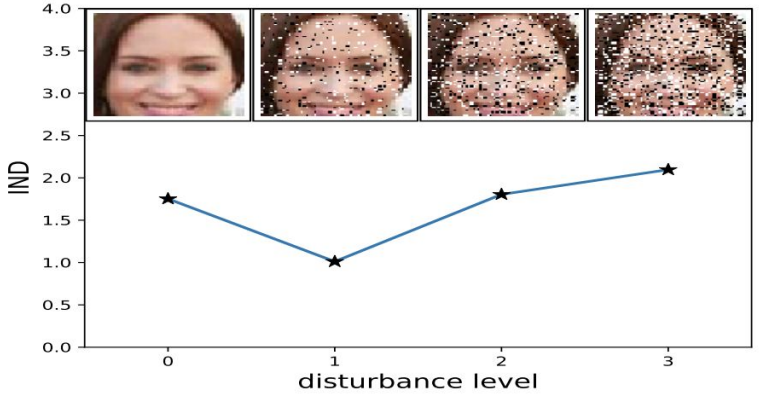


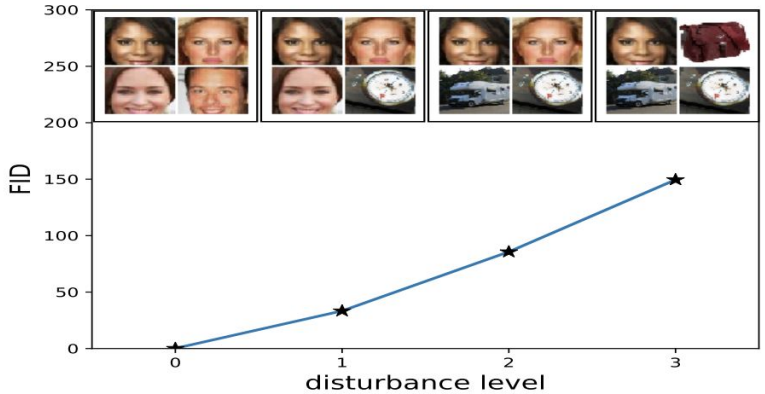


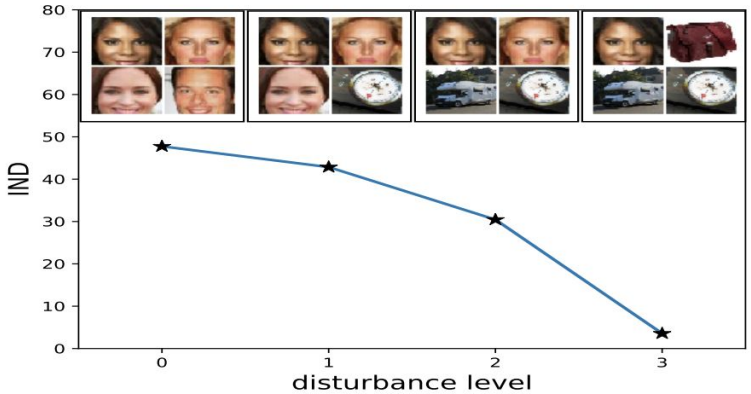
017)

Fréchet Inception Distance









17)

els

Generative Adversarial Networks

- Key pieces of GAN
  - Fast sampling
  - No inference
  - Goodfellow suggests building inference by reversing / inverting a GAN - not really shown to work so far
  - Notion of optimizing directly for what you care about - perceptual samples

Deep Convolutional GAN (DCGAN)

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

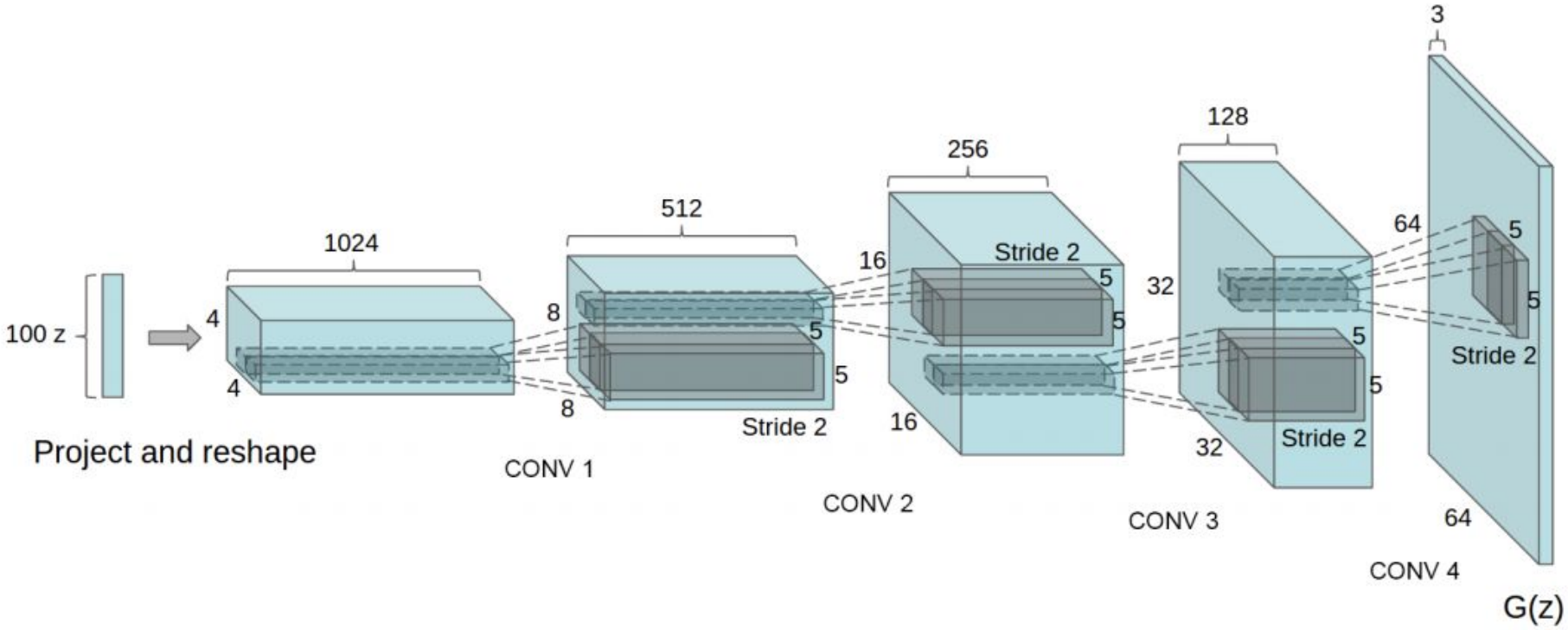
Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Deep Convolutional GAN (DCGAN)



DCGAN - Design Choices

- Architectures that worked well for supervised learning on images at scale weren't good for GANs.
- Got rid of spatial max pooling and used all-convolutional network design using strided convolutions.
- Barring first projection layer, every other layer in the generator is de-convolutional (transposed convolution with strides) for up-sampling.
- Batch Normalization helps in preventing mode collapse when training deep generators. BN NOT applied to generator output layer and discriminator input layer
- ReLU (instead of Maxout) for generator with tanh output
- Leaky ReLU for discriminator with sigmoid output
- Adam with a small step size (2e-4) and small momentum (0.5)
- Leaky ReLU slope of 0.2, batch size 128, inputs scaled to [-1, 1]

DCGAN - Key Results

- Good samples on datasets with 3M images (Faces, Bedrooms) for the first time

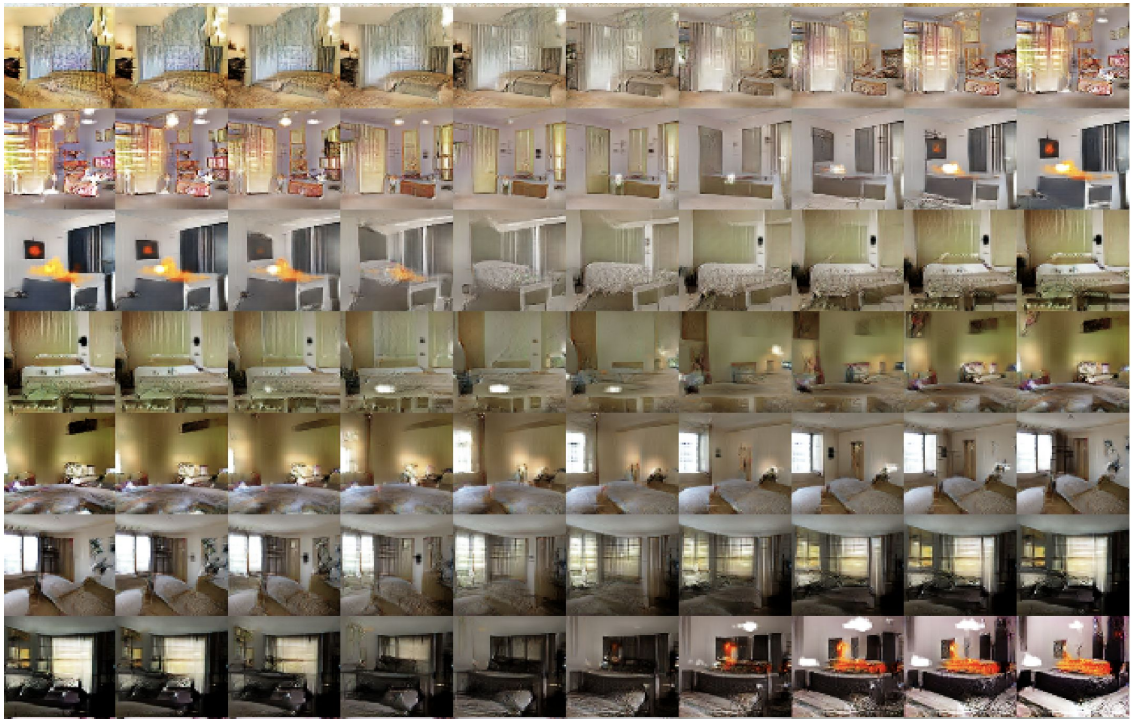
Radford et al 2016

DCGAN - Key Results

Radford et al 2016

DCGAN - Key Results

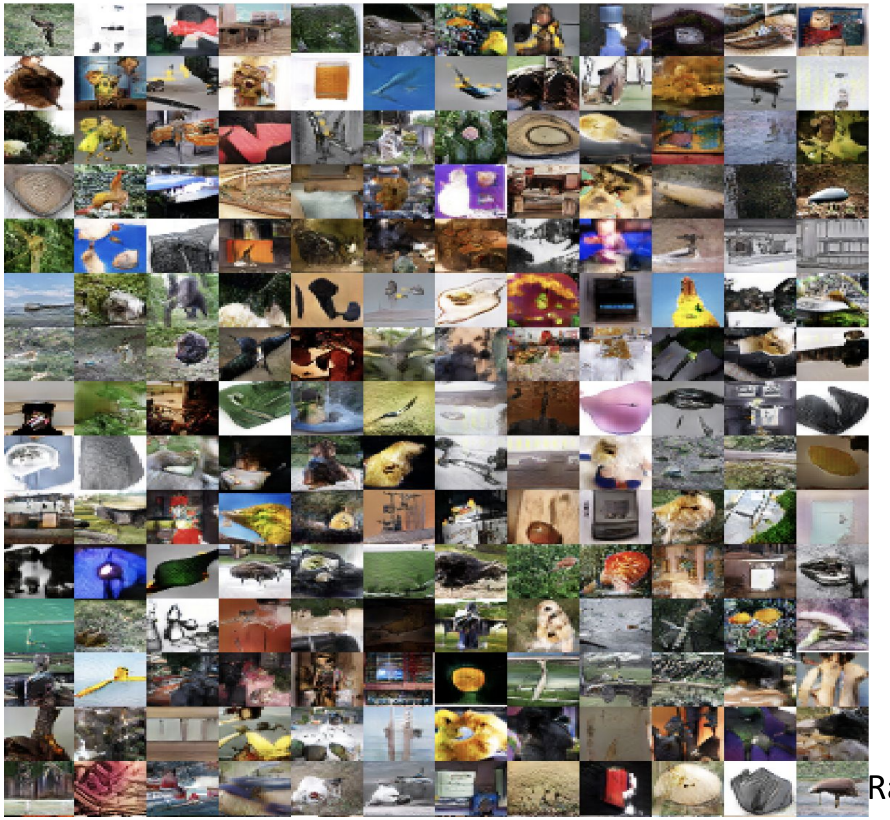
- Smooth interpolations in high dimensions



Radford et al 2016

DCGAN - Key Results

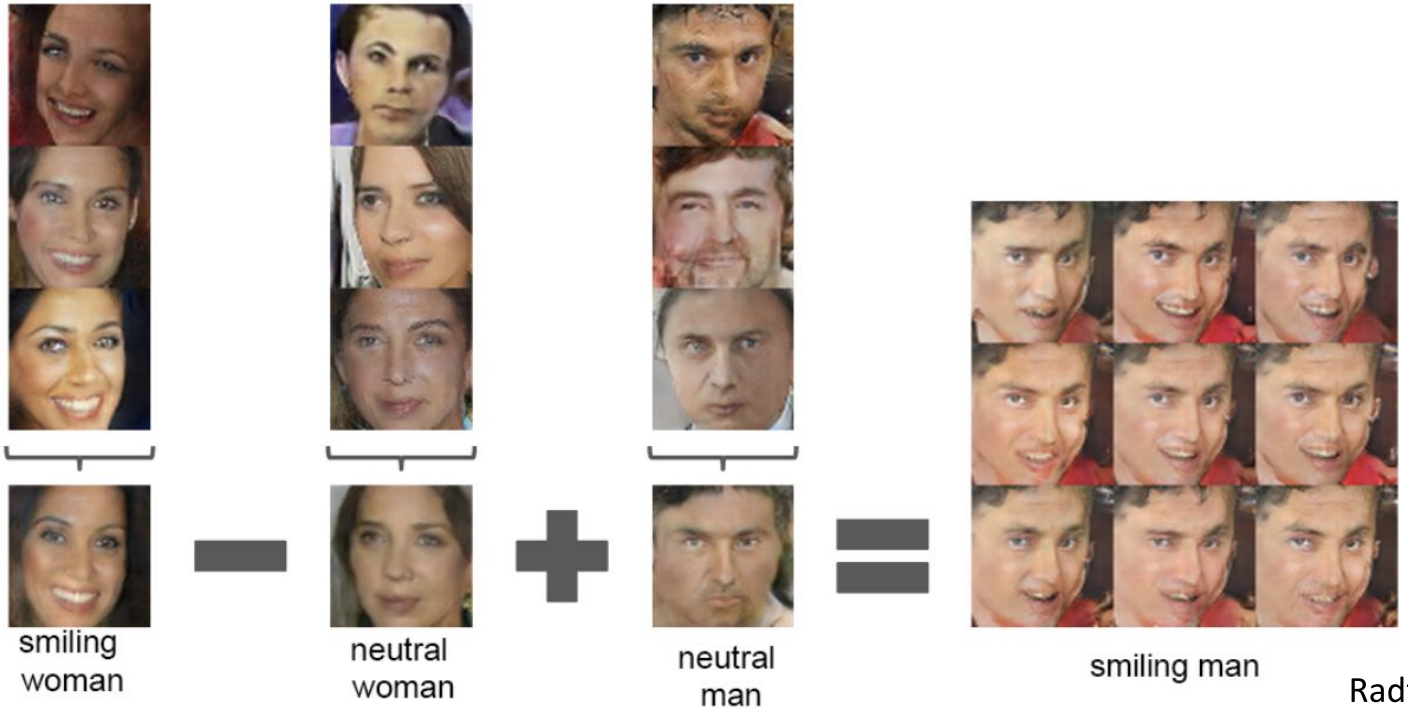
- Imagenet samples



Radford et al 2016

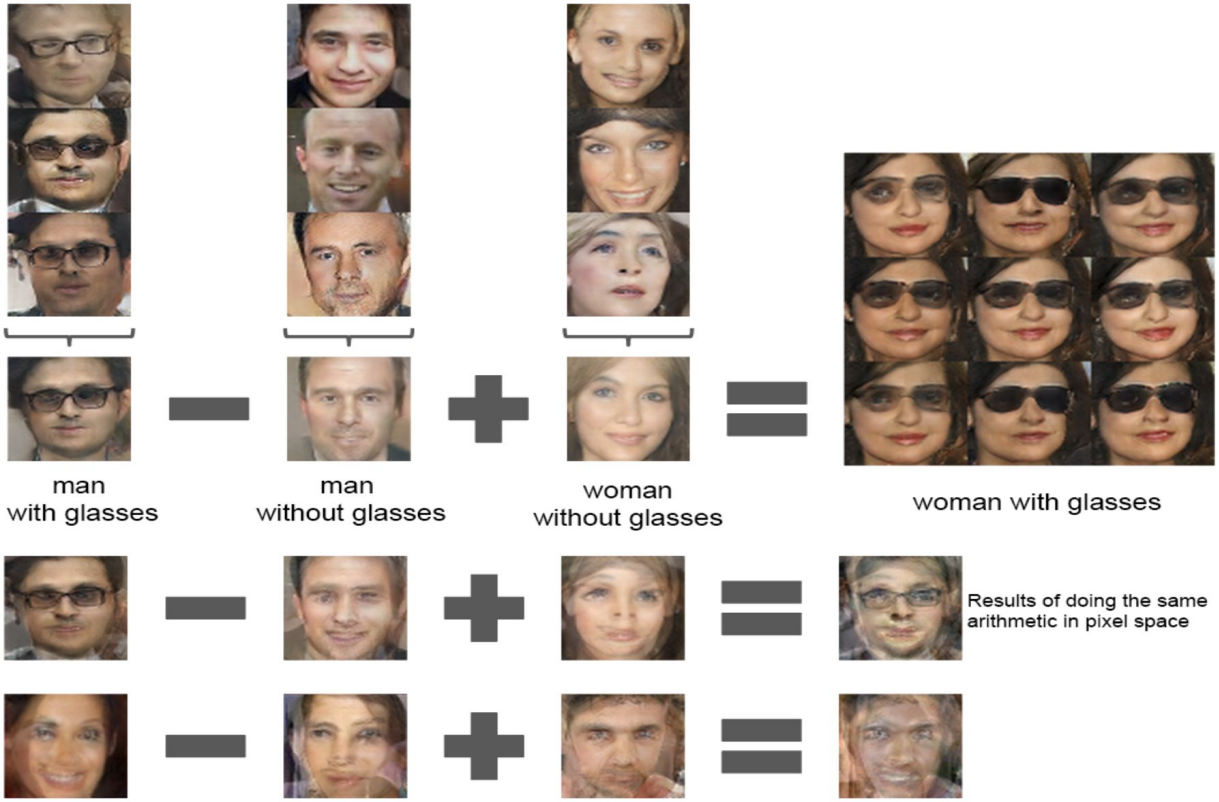
DCGAN - Key Results

- Vector Arithmetic



Radford et al 2016

DCGAN - Key Results



Radford et al 2016

DCGAN - Key Results

Radford et al 2016

DCGAN - Key Results

Representation Learning

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

Radford et al 2016

Issues even after DCGAN

- DCGAN proof that GANs can be “made” to work and produce really good (perceptually) samples and interpolations.
- But key issues remained
  - Unstable training
  - Brittle architectural choices
  - Not robust to architectures and hyperparameters

Improved training of GANs

- Feature Matching
- Historical Averaging
- Minibatch discrimination
- Virtual batch-normalization
- One-sided Label smoothing
- Semi-supervised Learning
- 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

Salimans 2016

Improved training of GANs

- Feature Matching

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

Generator objective

Salimans 2016

Improved training of GANs

- Historical Averaging

$$||\boldsymbol{\theta} - \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{\theta}[i]||^2$$

Salimans 2016

Improved training of GANs

- Minibatch discrimination

$$\mathbf{f}(\boldsymbol{x}_i) \in \mathbb{R}^A \qquad T \in \mathbb{R}^{A \times B \times C} \qquad M_i \in \mathbb{R}^{B \times C}$$

$$c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||M_{i,b} - M_{j,b}||_{L_1}) \in \mathbb{R}$$

$$o(\boldsymbol{x}_i)_b = \sum_{j=1}^{n} c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}$$

$$o(\boldsymbol{x}_i) = \left[o(\boldsymbol{x}_i)_1, o(\boldsymbol{x}_i)_2, \ldots, o(\boldsymbol{x}_i)_B\right] \in \mathbb{R}^B$$

$$o(\mathbf{X}) \in \mathbb{R}^{n \times B}$$

Salimans 2016

Allows to incorporate side information from other samples and is superior to feature matching in the unconditional setting. Helps addressing mode collapse by allowing discriminator to detect if the generated samples are too close to each other.

Improved training of GANs

- One-sided label smoothing

Default discriminator cost:

```
cross_entropy(1., discriminator(data))
+ cross_entropy(0., discriminator(samples))
```

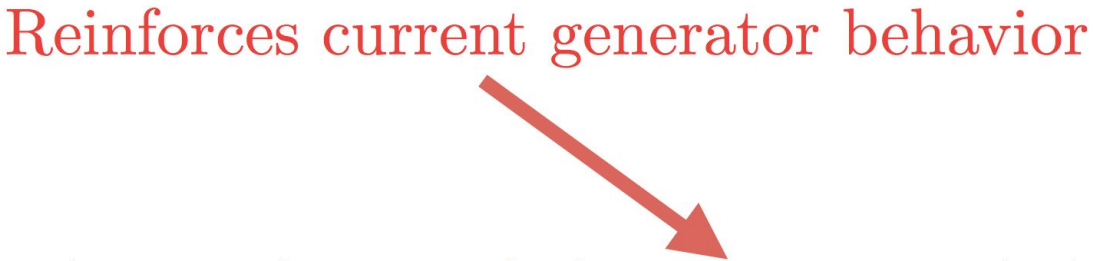

Goodfellow 2016

One-sided label smoothed cost (Salimans et al 2016):

```
cross_entropy(.9, discriminator(data))
+ cross_entropy(0., discriminator(samples))
```

Improved training of GANs

- Why one-sided?

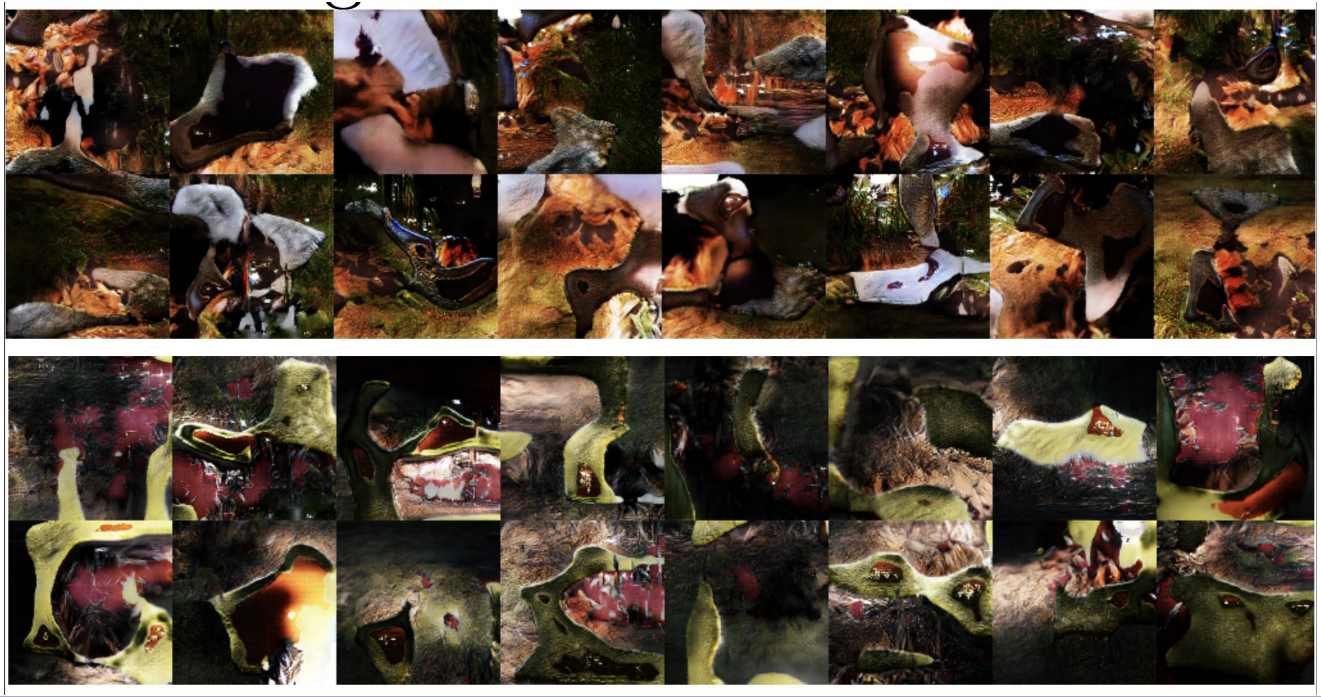


$$D(\boldsymbol{x}) = \frac{(1-\alpha)p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

Goodfellow 2016

Improved training of GANs

- Virtual Batch Normalization



Goodfellow 2016

Improved training of GANs

- Virtual Batch Normalization
  - Use a reference batch (fixed) to compute normalization statistics
  - Construct a batch containing the sample and reference batch

Improved training of GANs

- Semi-Supervised Learning
  - Predict labels in addition to fake/real in the discriminator
  - Approximate way of modeling p(x,y)
  - Generator doesn't have to made conditional p(x|y)
  - Use a deeper architecture for the discriminator compared to generator

$$L = -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x},y)}[\log p_{\text{model}}(y|\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim G}[\log p_{\text{model}}(y = K+1|\boldsymbol{x})]$$

$$= L_{\text{supervised}} + L_{\text{unsupervised}}, \text{ where}$$

$$L_{\text{supervised}} = -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x},y)} \log p_{\text{model}}(y|\boldsymbol{x}, y < K+1)$$

$$L_{\text{unsupervised}} = -\{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log[1 - p_{\text{model}}(y = K+1|\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim G} \log[p_{\text{model}}(y = K+1|\boldsymbol{x})]\}$$

Goodfellow 2016

Improved training of GANs

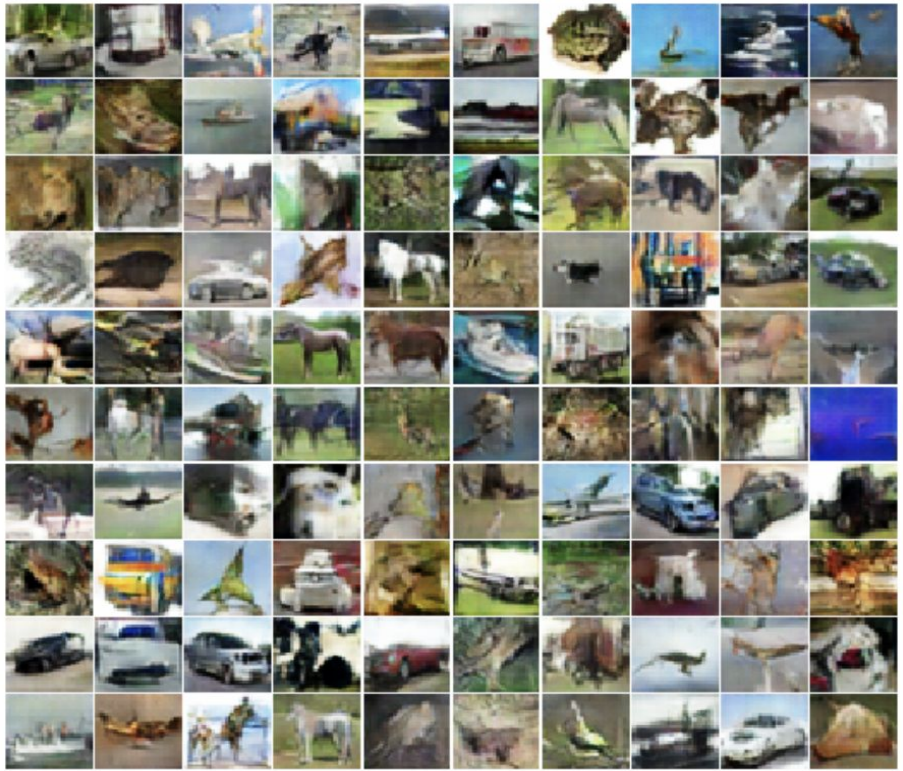
- Inception Score

$$\exp(\mathbb{E}_{\boldsymbol{x}}\mathbf{KL}(p(y|\boldsymbol{x})||p(y)))$$

  - Correlates with human judgement
  - Captures some necessity for diversity

Goodfellow 2016

Improved training of GANs



Salimans 2016

Wasserstein Distance

- We have seen $KL(P_r \| P_g)$ and $JSD(P_r, P_g) = KL(P_r \| (\frac{P_r + P_g}{2}) + KL(P_g \| (\frac{P_r + P_g}{2}))$

- Another distance measure inspired from Optimal Transport is the Earth Mover (EM) distance

- $$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[||x - y|| \right]$$

- Goal: Design a GAN objective function such that the generator minimizes the Earth Mover / Wasserstein distance between data and generated distributions.

Kantorovich Rubinstein Duality

- $W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[||x - y|| \right]$
- Intractable to estimate
- Kantorovich Rubinstein Duality: $W(P_r, P_g) = \sup_{\|f_L\| \leq 1} \mathbb{E}_{x\sim P_r} \left[f(x) \right] - \mathbb{E}_{x\sim P_g} \left[f(x) \right]$
- Search over joint distributions is now a search over 1-Lipschitz functions

Wasserstein GAN

Wasserstein GAN

Martin Arjovsky[1], Soumith Chintala[2], and Léon Bottou[1,2]

[1]Courant Institute of Mathematical Sciences
[2]Facebook AI Research

- Kantorovich Rubinstein Duality: $W(P_r, P_g) = \sup_{\|f_L\| \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_g}[f(x)]$

- Supremum over linear (function space) expectations => search over K-Lipschitz gives you K times the Wasserstein distance.

Wasserstein GAN

- $f : X \to Y$ is K-Lipschitz if for distance functions d_X and d_Y on X and Y $d_Y(f(x_1), f(x_2)) \leq K d_x(x_1, x_2)$
- Assume that we search over a parameterized family of functions f_w with $w \in \mathscr{W}$
- $\max_{w \in \mathscr{W}} \mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_g}[f_w(x)] \leq \sup_{\|f_L\| \leq K} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_g}[f(x)] = K \cdot W(P_r, P_g)$
- For P_g induced by $g_\theta(z)$ we can backprop through $\mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_g}[f_w(x)] : -\mathbb{E}_{z \sim p(z)}[\nabla_\theta f_w(g_\theta(z))]$

Wasserstein GAN - Pseudocode

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

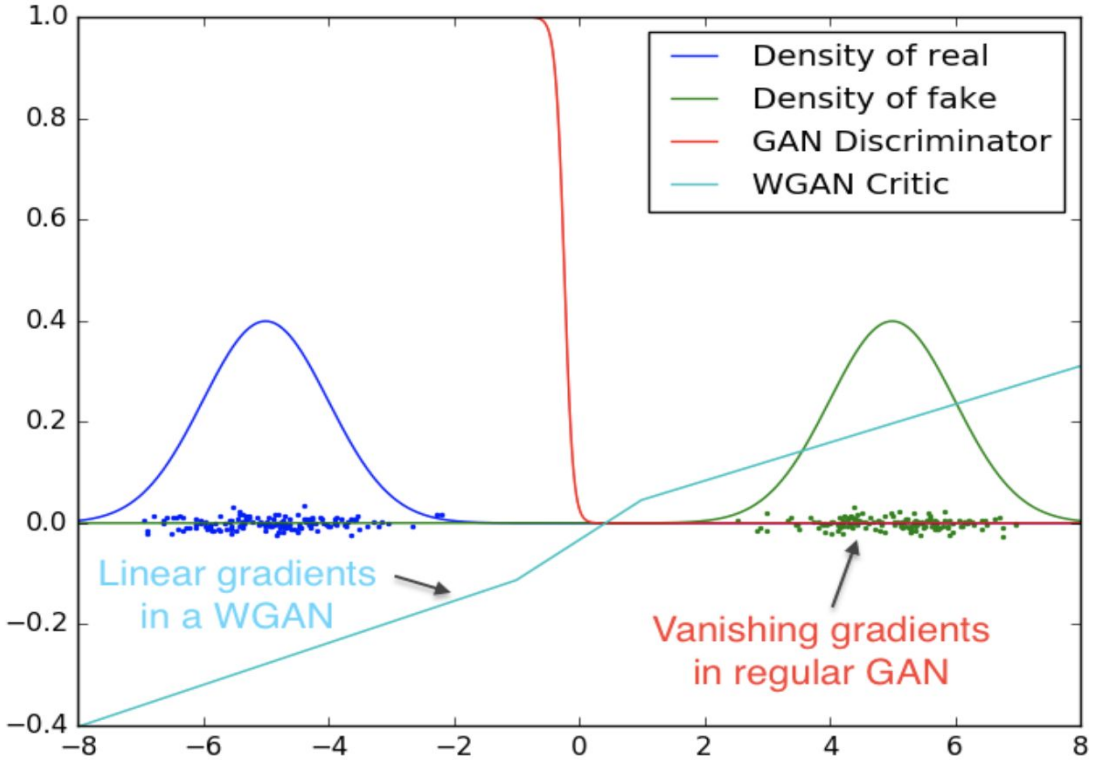
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

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

(Arjovsky et al 2017)

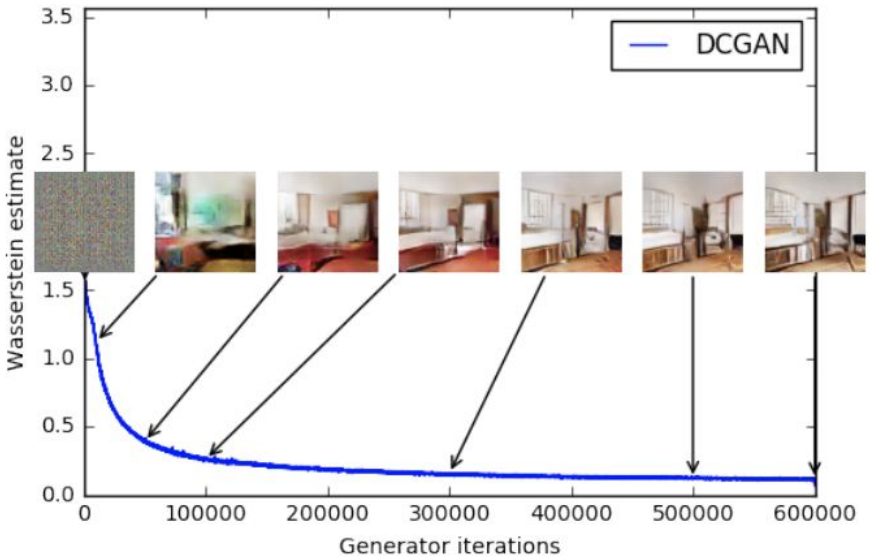
Wasserstein GAN - Training critic to converge



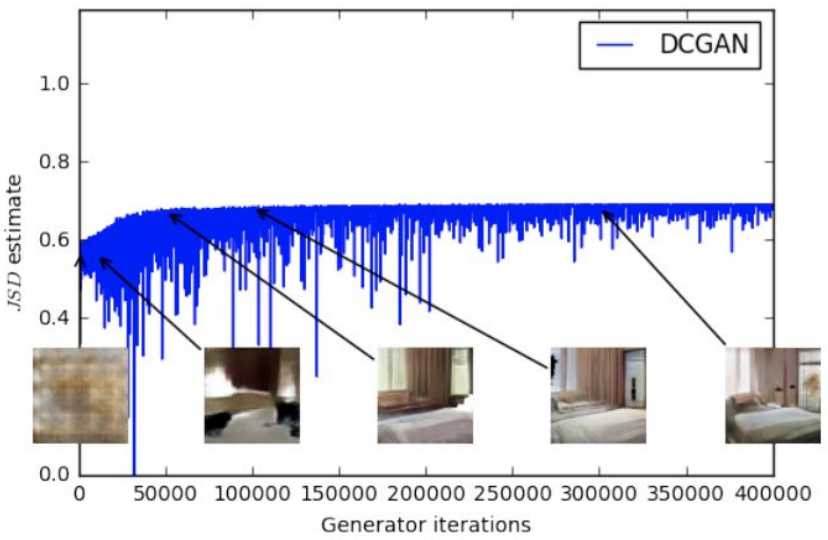
(Arjovsky et al 2017)

Wasserstein distance correlates with sample quality

Wasserstein Estimate

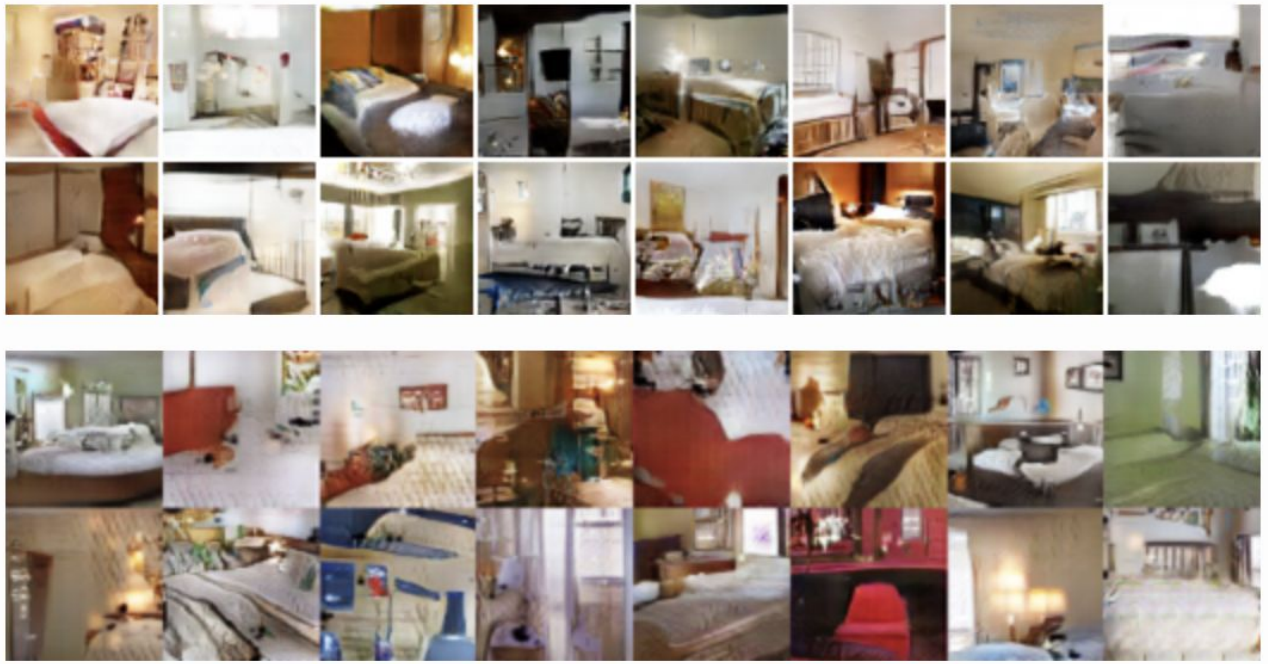


JSD Estimate



(Arjovsky et al 2017)

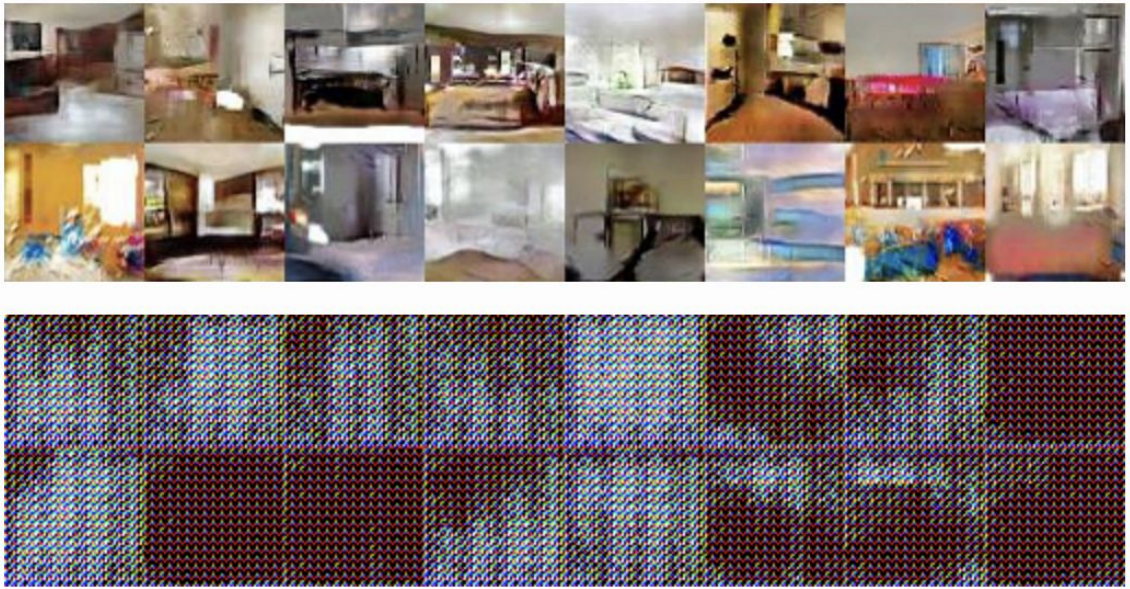
WGAN Samples on par with DCGAN



Top: WGAN with the same DCGAN architecture. *Bottom:* DCGAN

(Arjovsky et al 2017)

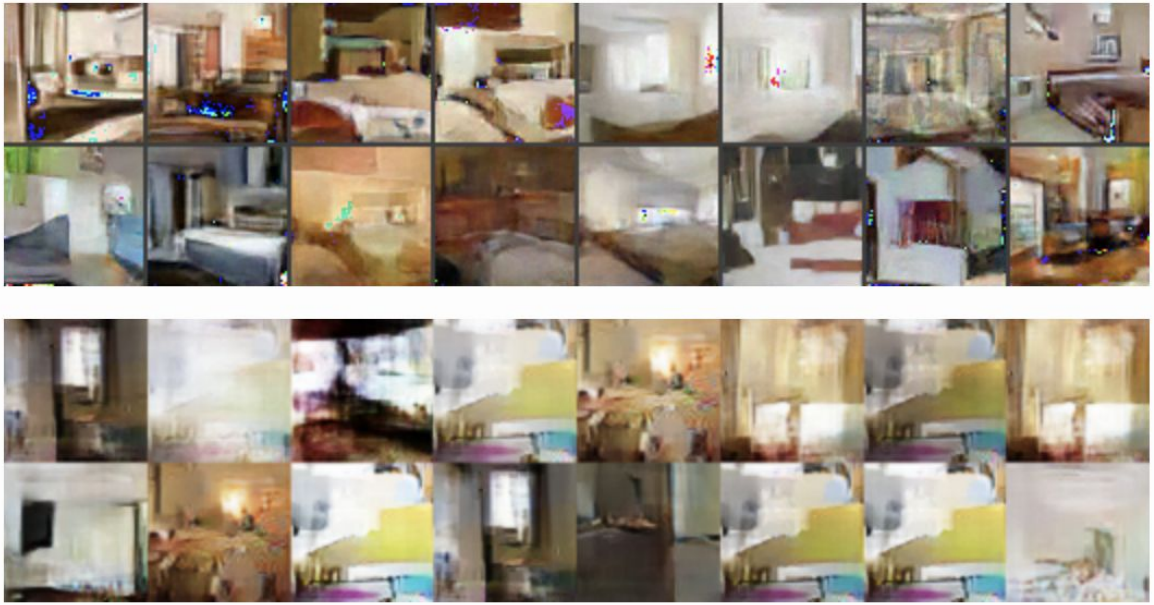
WGAN robust to architecture choices



Top: WGAN with DCGAN architecture, no batch norm. *Bottom:* DCGAN, no batch norm.

(Arjovsky et al 2017)

WGAN robust to architecture choices



Top: WGAN with MLP architecture. *Bottom:* Standard GAN, same architecture. (Arjovsky et al 2017)

WGAN Summary

Standard GAN

$$\min_G \max_D \mathbb{E}_{x \sim P_r} [\log D(x)] + \mathbb{E}_{\tilde{x} \sim P_g} [\log(1 - D(\tilde{x}))]$$

Wasserstein GAN

$$\min_G \max_{D \in \mathscr{D}} \mathbb{E}_{x \sim P_r} [D(x)] - \mathbb{E}_{\tilde{x} \sim P_g} [D(\tilde{x})]$$

(Arjovsky et al 2017)

WGAN Summary

- New divergence measure for optimizing the generator
- Addresses instabilities with JSD version (sigmoid cross entropy)
- Robust to architectural choices
- Progress on mode collapse and stability of derivative wrt input
- Introduces the idea of using lipschitzness to stabilize GAN training
- Negative:

> Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs). We experimented with simple variants (such as projecting the weights to a sphere) with little difference, and we stuck with weight clipping due to its simplicity and already good performance. However, we do leave the topic of enforcing Lipschitz constraints in a neural network setting for further investigation, and we actively encourage interested researchers to improve on this method.

(Arjovsky et al 2017)

WGAN-GP: Gradient Penalty for Lipschitzness

Improved Training of Wasserstein GANs

Ishaan Gulrajani[1]*, **Faruk Ahmed**[1], **Martin Arjovsky**[2], **Vincent Dumoulin**[1], **Aaron Courville**[1,3]
[1] Montreal Institute for Learning Algorithms
[2] Courant Institute of Mathematical Sciences
[3] CIFAR Fellow
igul222@gmail.com
{faruk.ahmed,vincent.dumoulin,aaron.courville}@umontreal.ca
ma4371@nyu.edu

Abstract

Generative Adversarial Networks (GANs) are powerful generative models, but suffer from training instability. The recently proposed Wasserstein GAN (WGAN) makes progress toward stable training of GANs, but sometimes can still generate only poor samples or fail to converge. We find that these problems are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to undesired behavior. We propose an alternative to clipping weights: penalize the norm of gradient of the critic with respect to its input. Our proposed method performs better than standard WGAN and enables stable training of a wide variety of GAN architectures with almost no hyperparameter tuning, including 101-layer ResNets and language models with continuous generators. We also achieve high quality generations on CIFAR-10 and LSUN bedrooms. †

Gulrajani et al 2017

WGAN-GP: Gradient Penalty for Lipschitzness

Proposition 1. *Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions in $\mathcal{X}$, a compact metric space. Then, there is a 1-Lipschitz function f^* which is the optimal solution of $\max_{\|f\|_L \leq 1} \mathbb{E}_{y \sim \mathbb{P}_r}[f(y)] - \mathbb{E}_{x \sim \mathbb{P}_g}[f(x)]$. Let π be the optimal coupling between $\mathbb{P}_r$ and $\mathbb{P}_g$, defined as the minimizer of: $W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\pi \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \pi}[\|x - y\|]$ where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of joint distributions $\pi(x, y)$ whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively. Then, if f^* is differentiable[‡], $\pi(x = y) = 0$[§], and $x_t = tx + (1-t)y$ with $0 \leq t \leq 1$, it holds that $\mathbb{P}_{(x,y) \sim \pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}\right] = 1$.*

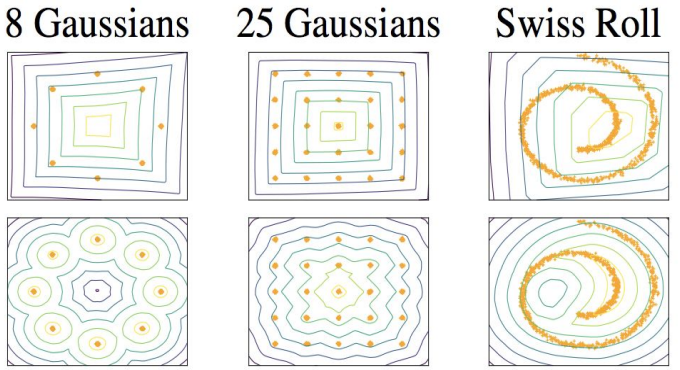
Corollary 1. *f^* has gradient norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$.*

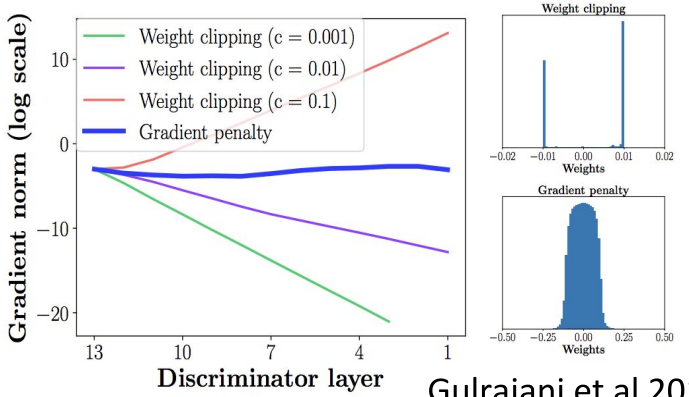
Gulrajani et al 2017

WGAN-GP: Gradient Penalty for Lipschitzness

$$\max_D \underbrace{\mathbb{E}_{x\sim P_r}\left[D(x)\right] - \mathbb{E}_{\tilde{x}\sim P_g}\left[D(\tilde{x})\right]}_{\text{Wasserstein critic objective}} + \lambda \underbrace{\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2\right]}_{\text{Gradient Penalty for Lipschitzness}}$$

$$\hat{x} \leftarrow \epsilon x + (1-\epsilon)\tilde{x}$$





Gulrajani et al 2017

WGAN-GP: Pseudocode

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

Gulrajani et al 2017

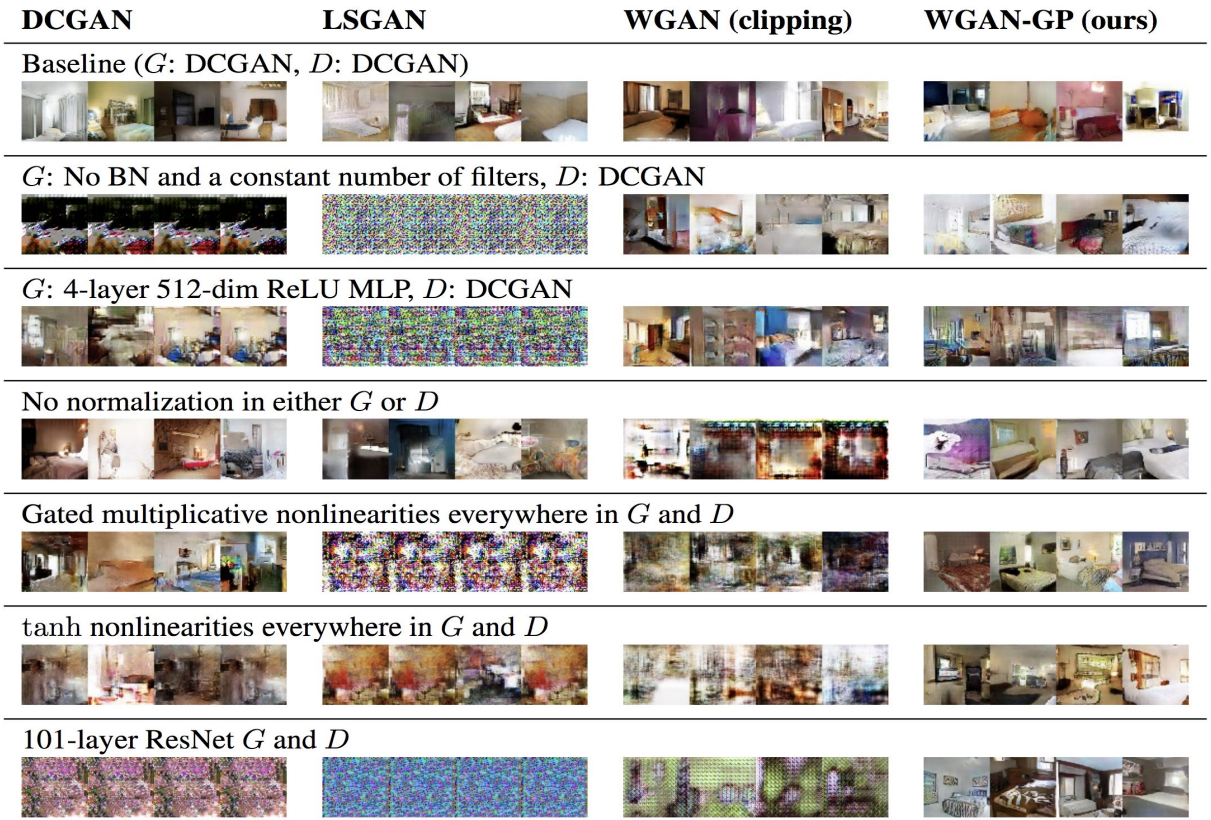
WGAN-GP: Robustness to architectures

Nonlinearity (G)	[ReLU, LeakyReLU, $\frac{\text{softplus}(2x+2)}{2} - 1$, tanh]
Nonlinearity (D)	[ReLU, LeakyReLU, $\frac{\text{softplus}(2x+2)}{2} - 1$, tanh]
Depth (G)	[4, 8, 12, 20]
Depth (D)	[4, 8, 12, 20]
Batch norm (G)	[True, False]
Batch norm (D; layer norm for WGAN-GP)	[True, False]
Base filter count (G)	[32, 64, 128]
Base filter count (D)	[32, 64, 128]

Min. score	Only GAN	Only WGAN-GP	Both succeeded	Both failed
1.0	0	8	192	0
3.0	1	88	110	1
5.0	0	147	42	11
7.0	1	104	5	90
9.0	0	0	0	200

Gulrajani et al 2017

WGAN-GP: Robustness to architectures



Gulrajani et al 2017

WGAN-GP: High quality samples

Gulrajani et al 2017

WGAN-GP: High quality samples

Table 3: Inception scores on CIFAR-10. Our unsupervised model achieves state-of-the-art performance, and our conditional model outperforms all others except SGAN.

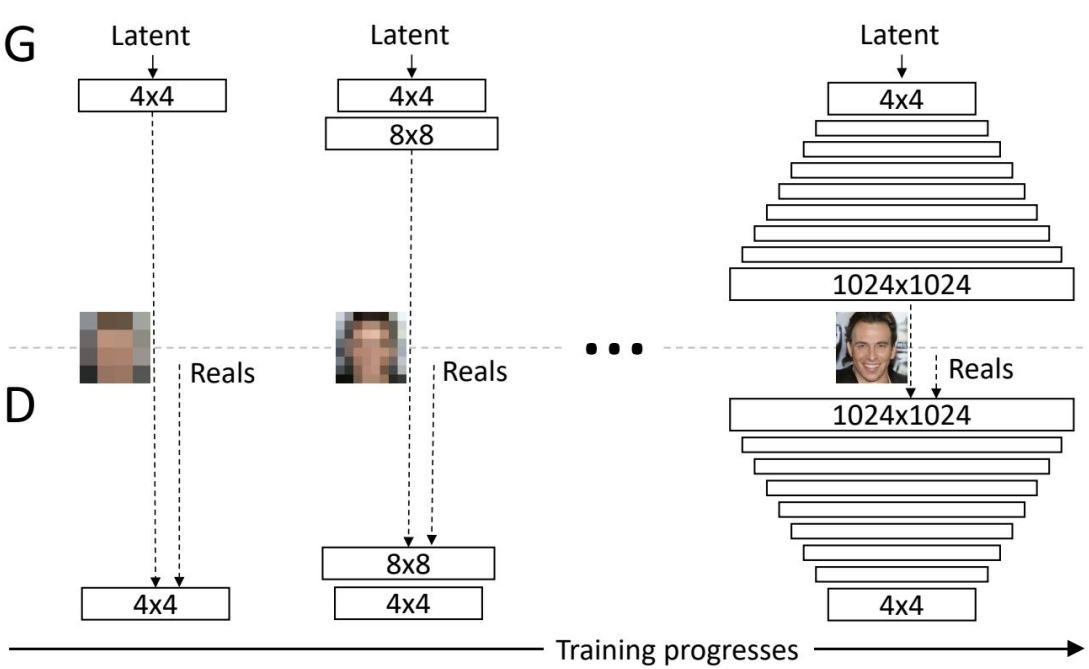
Unsupervised		Supervised	
Method	Score	Method	Score
ALI [8] (in [27])	$5.34 \pm .05$	SteinGAN [26]	6.35
BEGAN [4]	5.62	DCGAN (with labels, in [26])	6.58
DCGAN [22] (in [11])	$6.16 \pm .07$	Improved GAN [23]	$8.09 \pm .07$
Improved GAN (-L+HA) [23]	$6.86 \pm .06$	AC-GAN [20]	$8.25 \pm .07$
EGAN-Ent-VI [7]	$7.07 \pm .10$	SGAN-no-joint [11]	$8.37 \pm .08$
DFM [27]	$7.72 \pm .13$	WGAN-GP ResNet (ours)	$8.42 \pm .10$
WGAN-GP ResNet (ours)	$7.86 \pm .07$	**SGAN** [11]	$8.59 \pm .12$

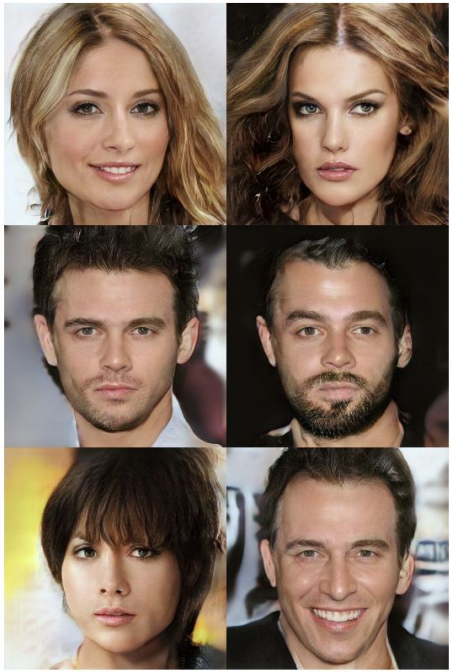
Gulrajani et al 2017

WGAN-GP: Summary

- Robustness to architectural choices.
- Robustness to hyperparameters
- Became a very popular GAN model - 1000+ citations, has been used in NVIDIA's Progressive GANs, StyleGAN, etc - biggest GAN successes
- Possible negative- slow wall clock time coz of gradient penalty?

Progressive growing of GANs



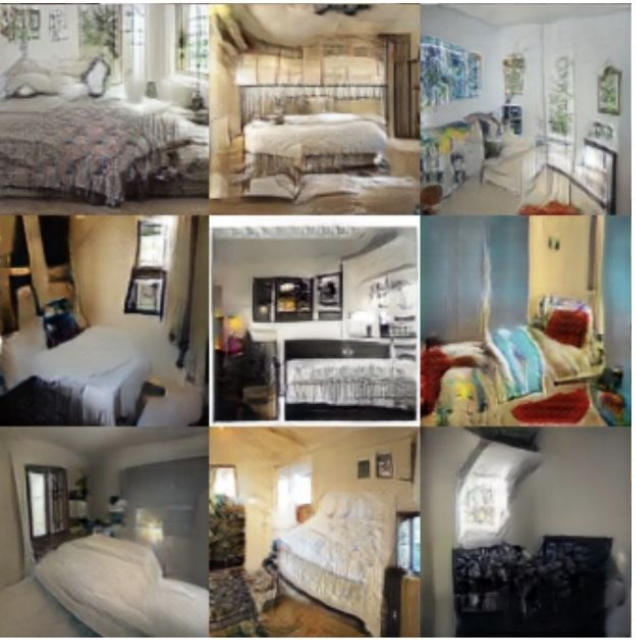


Karras et al 2017

Progressive growing of GANs

Karras et al 2017

Progressive growing of GANs



Mao et al. (2016b) (128 × 128)

Gulrajani et al. (2017) (128 × 128)

Our (256 × 256)

Karras et al 2017

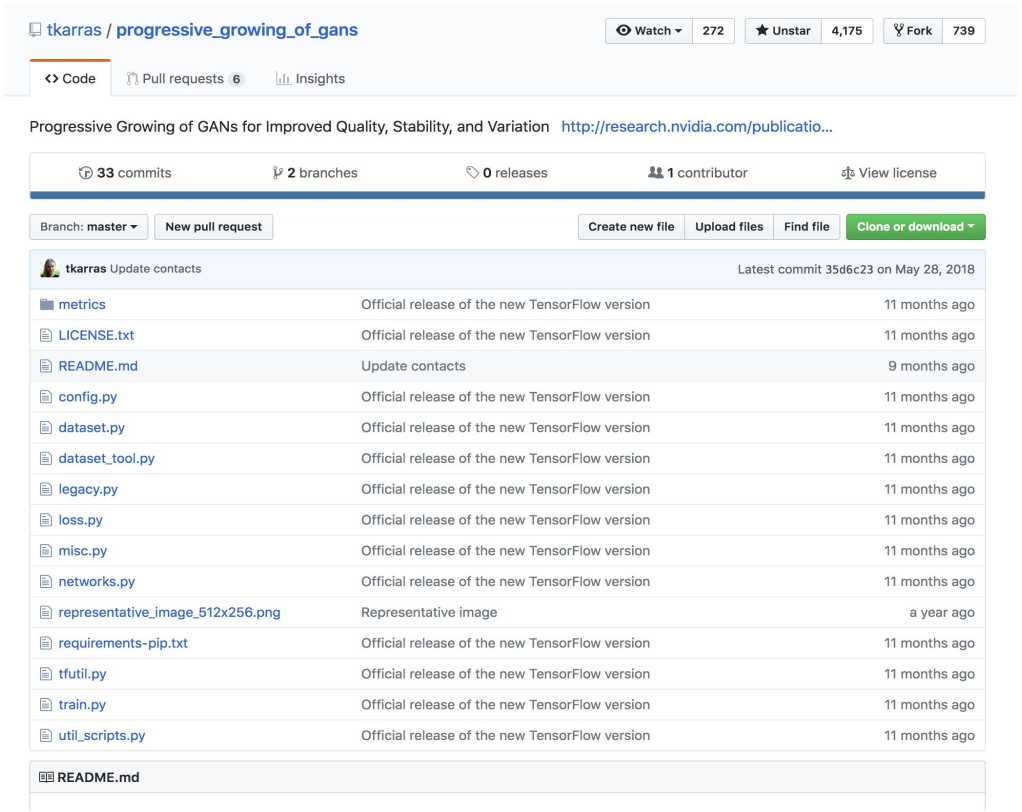
Progressive growing of GANs

Karras et al 2017

Progressive growing of GANs

Karras et al 2017

Progressive growing of GANs



tkarras / progressive_growing_of_gans

Watch 272 | Unstar 4,175 | Fork 739

Code | Pull requests 6 | Insights

Progressive Growing of GANs for Improved Quality, Stability, and Variation http://research.nvidia.com/publicatio...

33 commits | 2 branches | 0 releases | 1 contributor | View license

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

tkarras Update contacts — Latest commit 35d6c23 on May 28, 2018

metrics	Official release of the new TensorFlow version	11 months ago
LICENSE.txt	Official release of the new TensorFlow version	11 months ago
README.md	Update contacts	9 months ago
config.py	Official release of the new TensorFlow version	11 months ago
dataset.py	Official release of the new TensorFlow version	11 months ago
dataset_tool.py	Official release of the new TensorFlow version	11 months ago
legacy.py	Official release of the new TensorFlow version	11 months ago
loss.py	Official release of the new TensorFlow version	11 months ago
misc.py	Official release of the new TensorFlow version	11 months ago
networks.py	Official release of the new TensorFlow version	11 months ago
representative_image_512x256.png	Representative image	a year ago
requirements-pip.txt	Official release of the new TensorFlow version	11 months ago
tfutil.py	Official release of the new TensorFlow version	11 months ago
train.py	Official release of the new TensorFlow version	11 months ago
util_scripts.py	Official release of the new TensorFlow version	11 months ago

README.md

Spectral Normalization GAN (SNGAN)

SPECTRAL NORMALIZATION FOR GENERATIVE ADVERSARIAL NETWORKS

Takeru Miyato[1], Toshiki Kataoka[1], Masanori Koyama[2], Yuichi Yoshida[3]
{miyato, kataoka}@preferred.jp
koyama.masanori@gmail.com
yyoshida@nii.ac.jp
[1]Preferred Networks, Inc. [2]Ritsumeikan University [3]National Institute of Informatics

ABSTRACT

One of the challenges in the study of generative adversarial networks is the instability of its training. In this paper, we propose a novel weight normalization technique called spectral normalization to stabilize the training of the discriminator. Our new normalization technique is computationally light and easy to incorporate into existing implementations. We tested the efficacy of spectral normalization on CIFAR10, STL-10, and ILSVRC2012 dataset, and we experimentally confirmed that spectrally normalized GANs (SN-GANs) is capable of generating images of better or equal quality relative to the previous training stabilization techniques. The code with Chainer (Tokui et al., 2015), generated images and pretrained models are available at https://github.com/pfnet-research/sngan_projection.

Miyato et al 2017

Spectral Normalization GAN (SNGAN)

$$f(\boldsymbol{x}, \theta) = W^{L+1} a_L(W^L(a_{L-1}(W^{L-1}(\dots a_1(W^1 \boldsymbol{x}) \dots))))$$

$$D(\boldsymbol{x}, \theta) = \mathcal{A}(f(\boldsymbol{x}, \theta))$$

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

$$\arg\max_{\|f\|_{\text{Lip}} \leq K} V(G, D)$$

Miyato et al 2017

Spectral Normalization GAN (SNGAN)

- Key idea: Connecting Lipschitzness of discriminator to spectral norm of each layer.

$$g : \boldsymbol{h}_{in} \mapsto \boldsymbol{h}_{out}$$

$$\|g\|_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h}))$$

$$\sigma(A) := \max_{\boldsymbol{h}:\boldsymbol{h}\neq \mathbf{0}} \frac{\|A\boldsymbol{h}\|_2}{\|\boldsymbol{h}\|_2} = \max_{\|\boldsymbol{h}\|_2 \leq 1} \|A\boldsymbol{h}\|_2$$

Miyato et al 2017

Spectral Normalization GAN (SNGAN)

$$\|g\|_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h})) = \sup_{\boldsymbol{h}} \sigma(W) = \sigma(W)$$

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

$$\|f\|_{\text{Lip}} \leq \|(\boldsymbol{h}_L \mapsto W^{L+1}\boldsymbol{h}_L)\|_{\text{Lip}} \cdot \|a_L\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_{L-1} \mapsto W^{L}\boldsymbol{h}_{L-1})\|_{\text{Lip}}$$

$$\cdots \|a_1\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_0 \mapsto W^{1}\boldsymbol{h}_0)\|_{\text{Lip}} = \prod_{l=1}^{L+1} \|(\boldsymbol{h}_{l-1} \mapsto W^{l}\boldsymbol{h}_{l-1})\|_{\text{Lip}} = \prod_{l=1}^{L+1} \sigma(W^l)$$

$$\bar{W}_{\text{SN}}(W) := W/\sigma(W)$$

Miyato et al 2017

Spectral Normalization GAN (SNGAN)

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

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

Geometric GAN

Jae Hyun Lim[1], **Jong Chul Ye**[2,3]
[1] ETRI, South Korea
jaehyun.lim@etri.re.kr
[2] Dept. of Bio and Brain engineering, KAIST, South Korea
[3] Dept. of Mathematical Sciences, KAIST, South Korea
jong.ye@kaist.ac.kr

Miyato et al 2017

Spectral Normalization GAN (SNGAN)

$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
dense, $4 \times 4 \times 1024$
ResBlock up 1024
ResBlock up 512
ResBlock up 256
ResBlock up 128
ResBlock up 64
BN, ReLU, 3×3 conv 3
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
ResBlock down 64
ResBlock down 128
ResBlock down 256
ResBlock down 512
ResBlock down 1024
ResBlock 1024
ReLU
Global sum pooling
dense $\rightarrow 1$

(b) Discriminator for unconditional GANs.

RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
ResBlock down 64
ResBlock down 128
ResBlock down 256
Concat(Embed(y), $\boldsymbol{h}$)
ResBlock down 512
ResBlock down 1024
ResBlock 1024
ReLU
Global sum pooling
dense $\rightarrow 1$

(c) Discriminator for conditional GANs. For computational ease, we embedded the integer label $y \in \{0, \ldots, 1000\}$ into 128 dimension before concatenating the vector to the output of the intermediate layer.

Miyato et al 2017

Spectral Normalization GAN (SNGAN)

Miyato et al 2017

Spectral Normalization GAN (SNGAN)

```
# Batch Norm for convolution layers
def batch_normalization(x, *, eps=1e-8): # x in NHWC
  with tf.variable_scope('conv_bn'):
    mean, var = tf.nn.moments(x, [0, 1, 2], keep_dims=True)
    u = (x - mean)*tf.rsqrt(var + epsilon)
    scale, offset = tf.split(tf.get_variable('scale_offset',
                                [1, 1, 1, x.shape[-1].value] * 2))
    return u * scale + offset
```

Conditional Batch Normalization

```
# Conditional Batch Norm for convolution layers
def conditional_batch_normalization(x, y, *, eps=1e-8): # x in NHWC
  # x: 4D [b, h, w, c]; y: 2D [b, num_classes] (one-hot encoded)
  with tf.variable_scope('cond_conv_bn'):
    mean, var = tf.nn.moments(x, [0, 1, 2], keep_dims=True)
    u = (x - mean)*tf.rsqrt(var + epsilon)
    scale, offset = tf.split(
                    tf.layers.dense(y, 2 * x.shape[-1].value), 2, -1)
    return u * scale + offset
```

Conditional Batch Normalization with spec-norm

```
# Conditional Batch Norm (with SN) for convolution layers
def conditional_batch_normalization(x, y, *, eps=1e-8): # x in NHWC
  # x: 4D [b, h, w, c]; y: 2D [b, num_classes] (one-hot encoded)
  with tf.variable_scope('cond_conv_bn'):
    mean, var = tf.nn.moments(x, [0, 1, 2], keep_dims=True)
    u = (x - mean)*tf.rsqrt(var + epsilon)
    scale, offset = tf.split(
                    dense(y, 2 * x.shape[-1].value, use_sn=True), 2, -1)
    return u * scale + offset
```

Power Iteration to implement spec-norm

```
# Power Iteration
def spec_norm(w_, *, n_iterations=1):
  w = tf.reshape(w_, [-1, w.shape[-1]])
  u = tf.get_variable('u', shape=(w.shape[0], 1), trainable=False)
  for _ in range(n_iterations):
    v = tf.nn.l2_normalize(tf.matmul(w, u, transpose_a=True), epsilon)
    u = tf.nn.l2_normalize(tf.matmul(w, v), epsilon)
  u, v = tf.stop_gradient(u), tf.stop_gradient(v)
  norm_value = tf.matmul(tf.matmul(tf.transpose(u), w), v)
  w_normalized = tf.reshape(w / norm_value, w_.shape)
  return w_normalized
```

Power Iteration to implement spec-norm

```
# Power Iteration
def dense(x, *, n_out, name='dense', sn_iters=1, use_sn=True):
  with tf.get_variable(name):
    w = tf.get_variable('w', [x.shape[-1].value, n_out])
    if use_sn:
      w = spec_norm(w, n_iterations=sn_iters)
    b = tf.get_variable('b', [n_out], initializer=tf.zeros_initializer())
    return tf.nn.bias_add(tf.matmul(x, w), b)
```

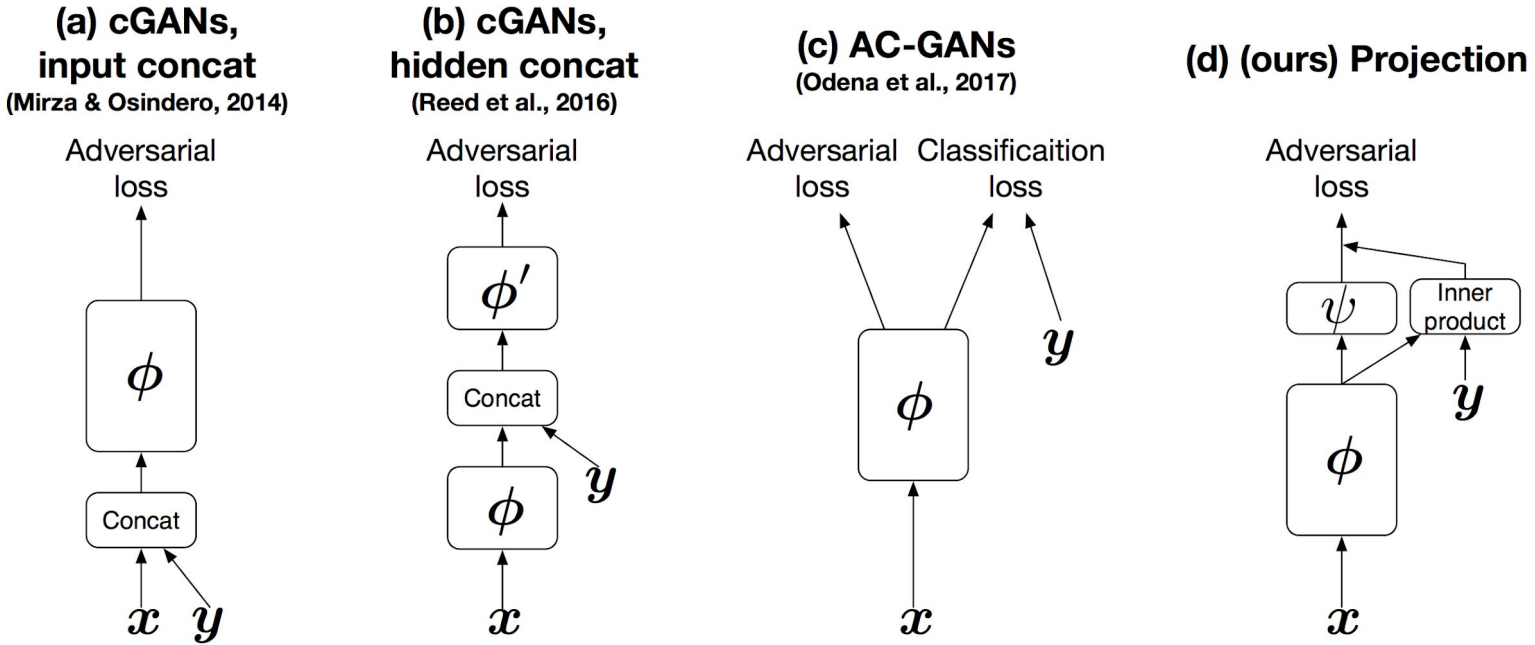
Power Iteration to implement spec-norm

```
# Power Iteration
def conv2d(x, *, n_out, kernels, strides, name, sn_iters=1, use_sn=True):
  with tf.get_variable(name):
    w = tf.get_variable('w', [*kernels, x.shape[-1].value, n_out])
    if use_sn:
      w = spec_norm(w, n_iterations=sn_iters)
    b = tf.get_variable('b', [n_out], initializer=tf.zeros_initializer())
    return tf.nn.bias_add(tf.nn.conv2d(x, w, [1, *strides, 1])
```

SNGAN: Summary

- High quality samples for class conditional GAN at Imagenet scale
- First GAN to work on full Imagenet (million samples)
- Computational benefits over WGAN-GP

Projection Discriminator



Self Attention GAN (SAGAN)

Self-Attention Generative Adversarial Networks

Han Zhang* Rutgers University

Ian Goodfellow Google Brain

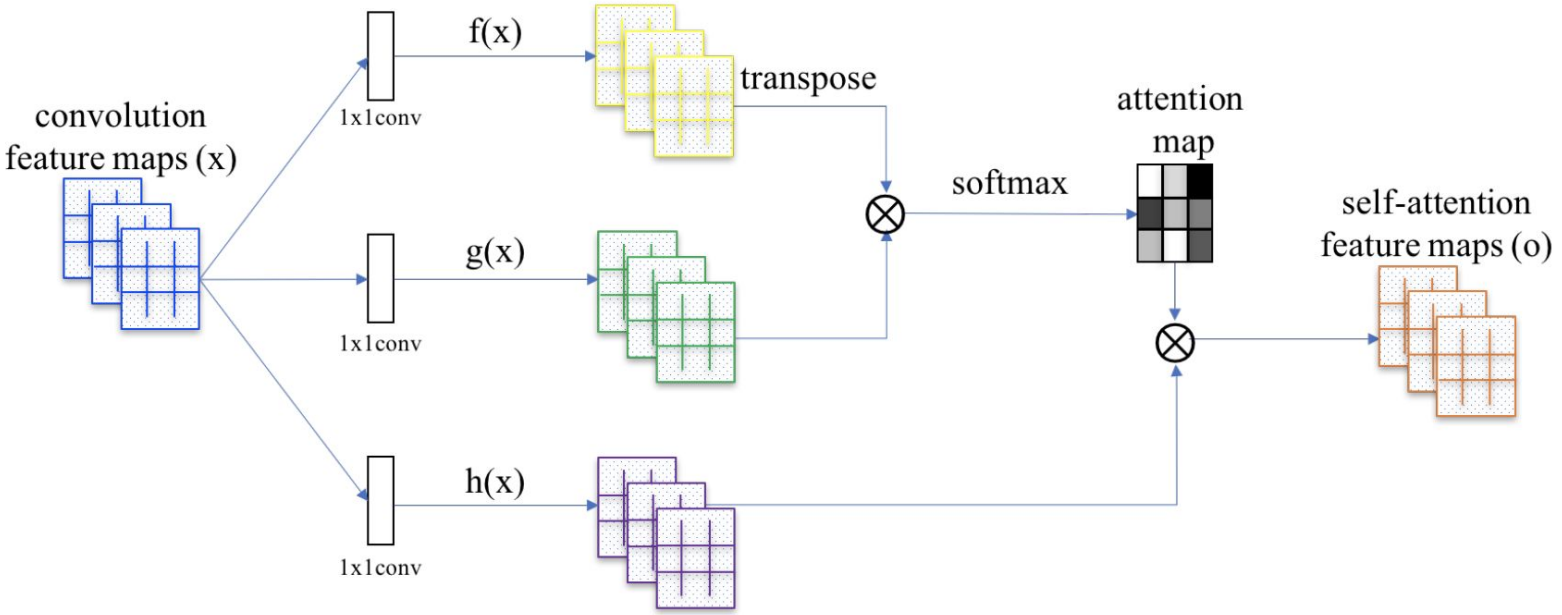
Dimitris Metaxas Rutgers University

Augustus Odena Google Brain

Abstract

In this paper, we propose the Self-Attention Generative Adversarial Network (SAGAN) which allows attention-driven, long-range dependency modeling for image generation tasks. Traditional convolutional GANs generate high-resolution details as a function of only spatially local points in lower-resolution feature maps. In SAGAN, details can be generated using cues from all feature locations. Moreover, the discriminator can check that highly detailed features in distant portions of the image are consistent with each other. Furthermore, recent work has shown that generator conditioning affects GAN performance. Leveraging this insight, we apply spectral normalization to the GAN generator and find that this improves training dynamics. The proposed SAGAN achieves the state-of-the-art results, boosting the best published Inception score from 36.8 to 52.52 and reducing Fréchet Inception distance from 27.62 to 18.65 on the challenging ImageNet dataset. Visualization of the attention layers shows that the generator leverages neighborhoods that correspond to object shapes rather than local regions of fixed shape.

Self Attention GAN (SAGAN)



Zhang et al 2018

Self Attention GAN (SAGAN)

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W_f x},\ \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{W_g x}$$

$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})}$$

$$\boldsymbol{s_{ij}} = \boldsymbol{f}(\boldsymbol{x_i})^T \boldsymbol{g}(\boldsymbol{x_j})$$

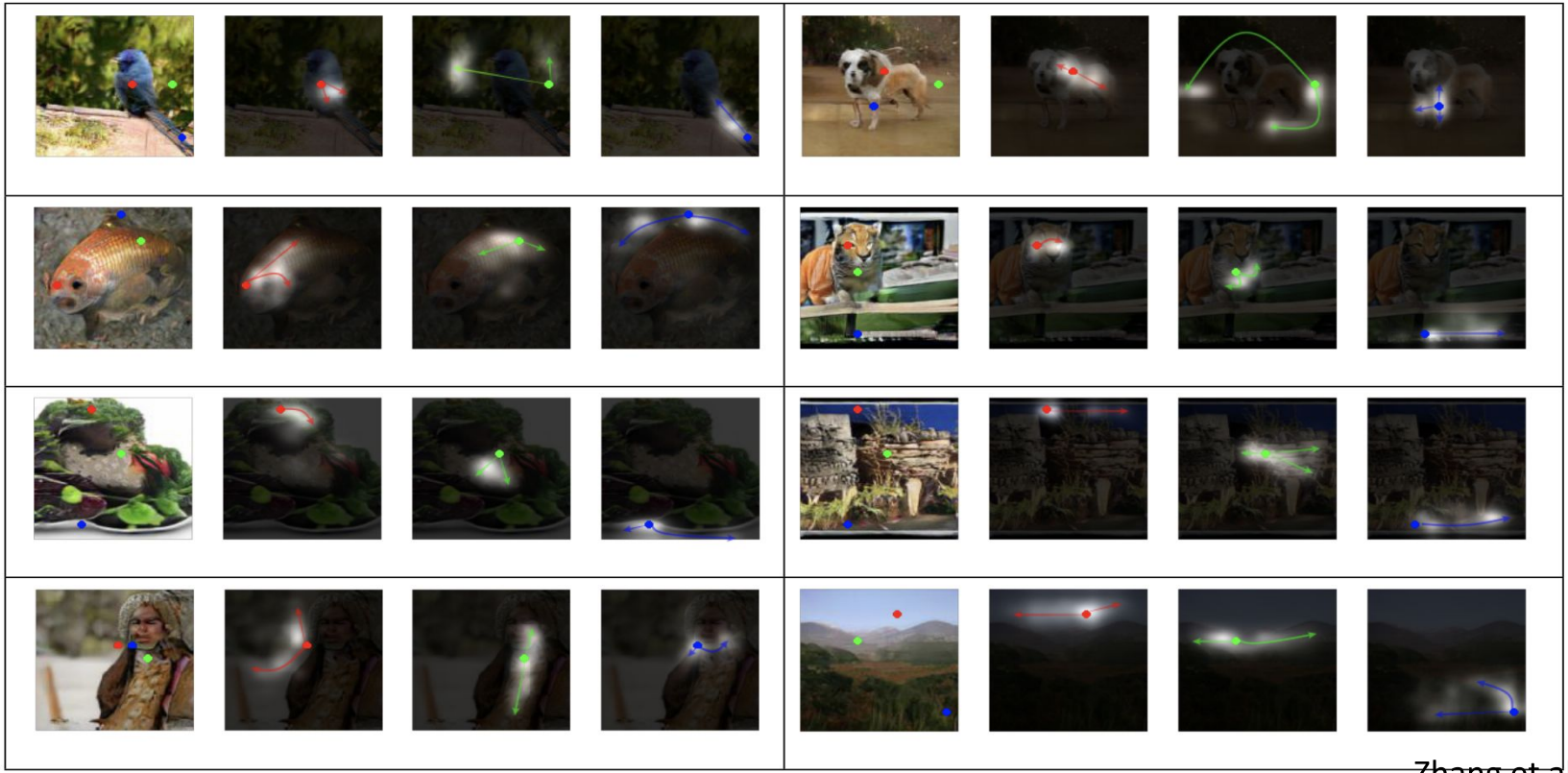
$$\boldsymbol{y_i} = \gamma \boldsymbol{o_i} + \boldsymbol{x_i}$$

Zhang et al 2018

Self Attention GAN (SAGAN)

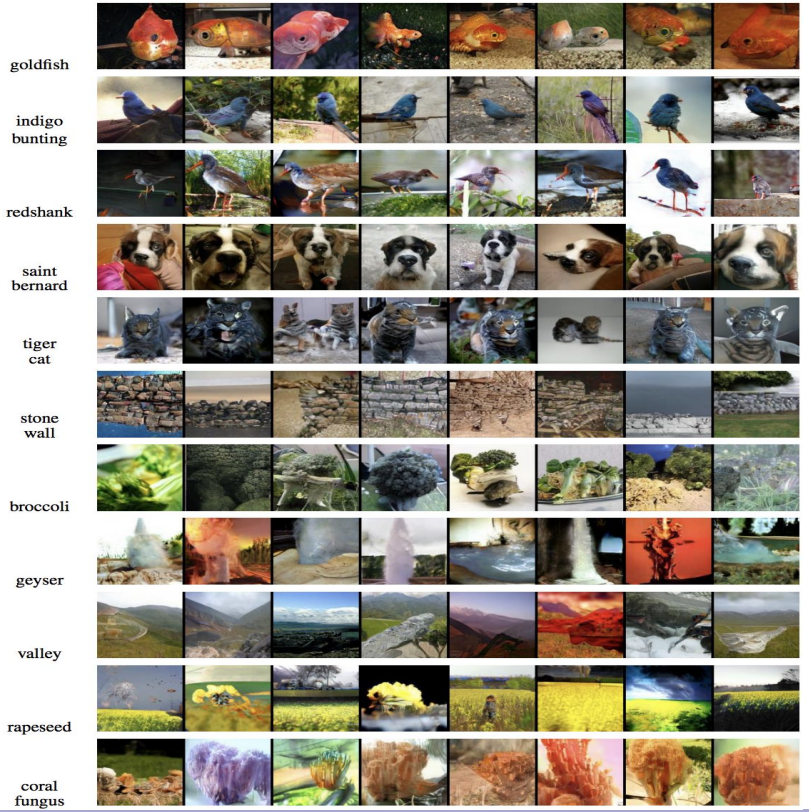
- Salient bits:
  - Applies spectral normalization to both the generator and discriminator weight matrices
    - This is counter-intuitive to popular belief that you only have to weaken or mathematically condition (regularize) the discriminator
  - Uses self-attention in both the generator and discriminator
    - Architectural upgrade
  - Hinge Loss
    - Becoming well adopted now
  - First GAN to produce “good” unconditional full Imagenet samples
    - Caveat: Not ablated for other choices
  - Conditional models
    - CBN for G, Projection Discriminator for D

Self Attention GAN (SAGAN)



Zhang et al 2018

Self Attention GAN (SAGAN)



Zhang et al 2018

Self Attention GAN (SAGAN)

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

Table 2: Comparison of the proposed SAGAN with state-of-the-art GAN models [19, 17] for class conditional image generation on ImageNet. FID of SNGAN-projection is calculated from officially released weights.

Zhang et al 2018

BigGAN

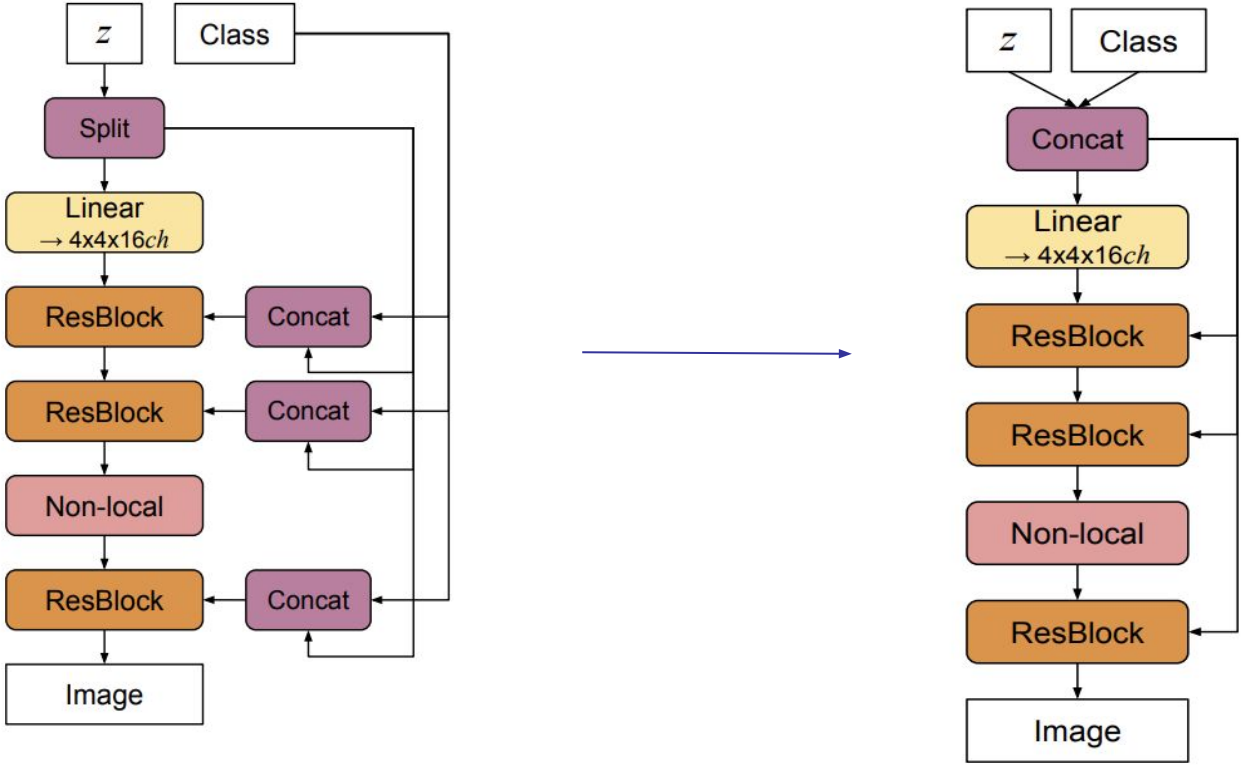
Big-GAN

BigGAN

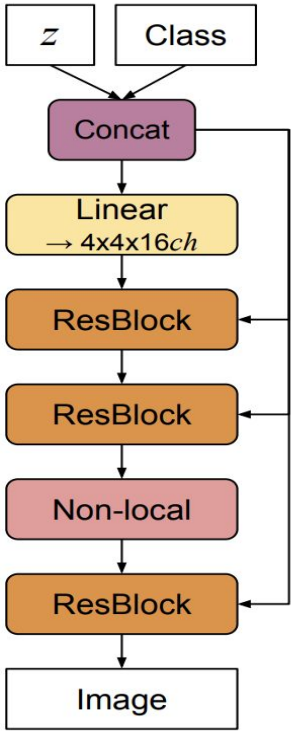
$$R_\beta(W) = \beta \|W^\top W - I\|_{\mathrm{F}}^2$$

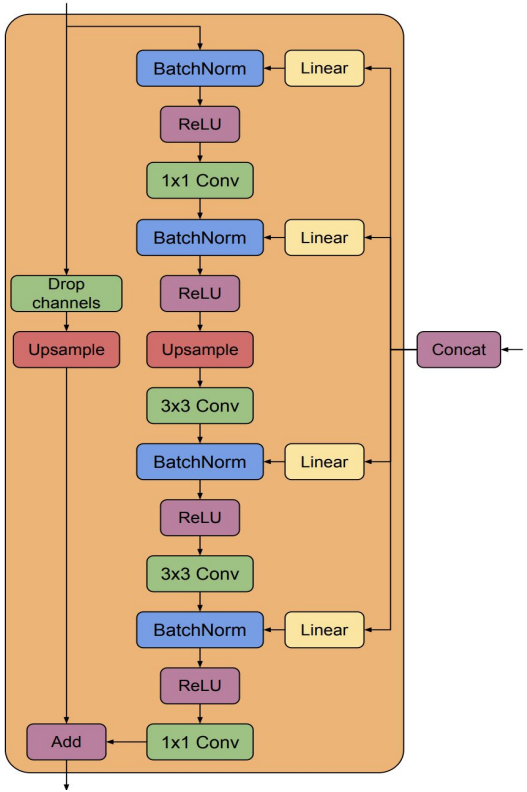
$$R_\beta(W) = \beta \|W^\top W \odot (\mathbf{1} - I)\|_{\mathrm{F}}^2,$$

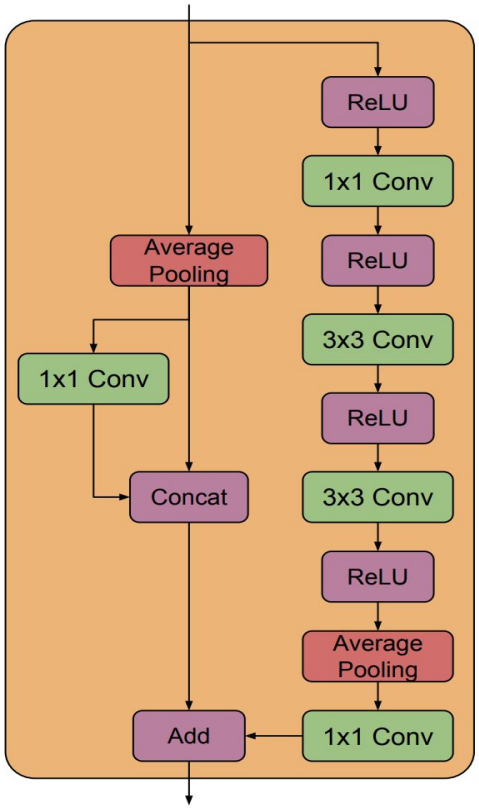
BigGAN and BigGAN-deep



BigGAN


z
Class
Concat
Linear
→ 4x4x16ch
ResBlock
ResBlock
Non-local
ResBlock
Image


BatchNorm
Linear
ReLU
1x1 Conv
BatchNorm
Linear
Drop channels
ReLU
Upsample
Upsample
Concat
3x3 Conv
BatchNorm
Linear
ReLU
3x3 Conv
BatchNorm
Linear
ReLU
Add
1x1 Conv


ReLU
1x1 Conv
Average Pooling
ReLU
3x3 Conv
1x1 Conv
ReLU
Concat
3x3 Conv
ReLU
Average Pooling
Add
1x1 Conv

BigGAN

Table 6: BigGAN architecture for 512×512 images. Relative to the 256×256 architecture, we add an additional ResBlock at the 512×512 resolution. Memory constraints force us to move the non-local block in both networks back to 64×64 resolution as in the 128×128 pixel setting.

$z \in \mathbb{R}^{160} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(20 + 128) \to 4 \times 4 \times 16ch$
ResBlock up $16ch \to 16ch$
ResBlock up $16ch \to 8ch$
ResBlock up $8ch \to 8ch$
ResBlock up $8ch \to 4ch$
Non-Local Block (64×64)
ResBlock up $4ch \to 2ch$
ResBlock up $2ch \to ch$
ResBlock up $ch \to ch$
BN, ReLU, 3×3 Conv $ch \to 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{512 \times 512 \times 3}$
ResBlock down $ch \to ch$
ResBlock down $ch \to 2ch$
ResBlock down $2ch \to 4ch$
Non-Local Block (64×64)
ResBlock down $4ch \to 8ch$
ResBlock down $8ch \to 8ch$
ResBlock down $8ch \to 16ch$
ResBlock down $16ch \to 16ch$
ResBlock $16ch \to 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\to 1$)

(b) Discriminator

BigGAN

- Salient bits
  - Increase your batch size (as much as you can)
  - Increase your model size
  - Wider helps as much as deeper
  - Fuse class information at all levels
  - Hinge Loss
  - Orthonormal regularization & Truncation Trick

Big-GAN

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	$8.7 \pm .6/98.8 \pm 3$	$7.7 \pm .2/126.5 \pm 0$	$9.6 \pm .4/166.3 \pm 1$	$25 \pm 2/206 \pm 2$
BigGAN	256	$8.7 \pm .1/142.3 \pm 2$	$7.7 \pm .1/178.0 \pm 5$	$9.3 \pm .3/233.1 \pm 1$	$25 \pm 5/291 \pm 4$
BigGAN	512	$8.1/144.2$	$7.6/170.3$	$11.8/241.4$	$27.0/275$
BigGAN-deep	128	$5.7 \pm .3/124.5 \pm 2$	$6.3 \pm .3/148.1 \pm 4$	$7.4 \pm .6/166.5 \pm 1$	$25 \pm 2/253 \pm 11$
BigGAN-deep	256	$6.9 \pm .2/171.4 \pm 2$	$7.0 \pm .1/202.6 \pm 2$	$8.1 \pm .1/232.5 \pm 2$	$27 \pm 8/317 \pm 6$
BigGAN-deep	512	$7.5/152.8$	$7.7/181.4$	$11.5/241.5$	$39.7/298$

BigGAN

Batch	Ch.	Param (M)	Shared	Skip-z	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($\pm$1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03($\pm$1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85($\pm$3.83)
2048	96	173.5	✗	✗	✗	295($\pm$18)	9.54($\pm$0.62)	92.98($\pm$4.27)
2048	96	160.6	✓	✗	✗	185($\pm$11)	9.18($\pm$0.13)	94.94($\pm$1.32)
2048	96	158.3	✓	✓	✗	152($\pm$7)	8.73($\pm$0.45)	98.76($\pm$2.84)
2048	96	158.3	✓	✓	✓	165($\pm$13)	8.51($\pm$0.32)	99.31($\pm$2.10)
2048	64	71.3	✓	✓	✓	371($\pm$7)	10.48($\pm$0.10)	86.90($\pm$0.61)

BigGAN - Truncation Trick

(a)

(b)

Big-GAN

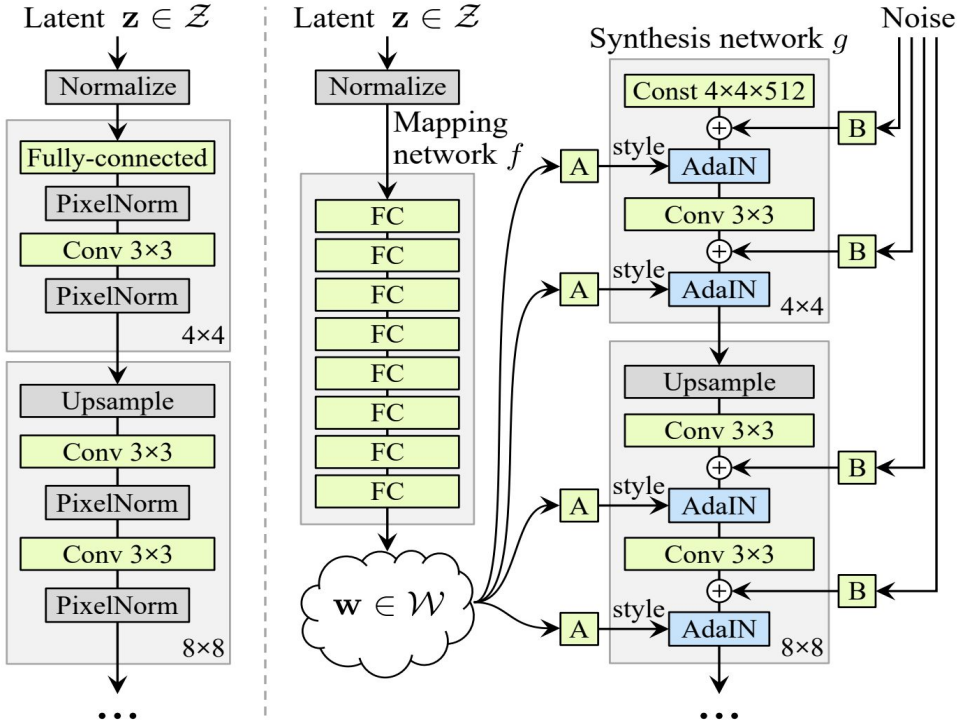
(a) 128×128

(b) 256×256

(c) 512×512

(d)

Style GAN



(a) Traditional (b) Style-based generator

Style GAN