ML Engineer 과정

Generative Adversarial Networks

Gunhee Kim

Computer Science and Engineering

서 울 대 학 교

SEOUL NATIONAL UNIVERSITY

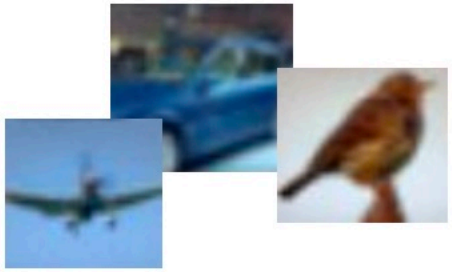
Outline

- GANs
- Recent Advances in GANs

Generative Models

Given training data, generate new samples from the same distribution

- Given training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, which are assumed to be sampled from true data distribution $P(\boldsymbol{x})$
- Want to train so that model distribution $\hat{P}(\boldsymbol{x}) \sim P(\boldsymbol{x})$ as possible
- Can address density estimation, a core problem in unsupervised learning



Training data $\sim P(\boldsymbol{x})$

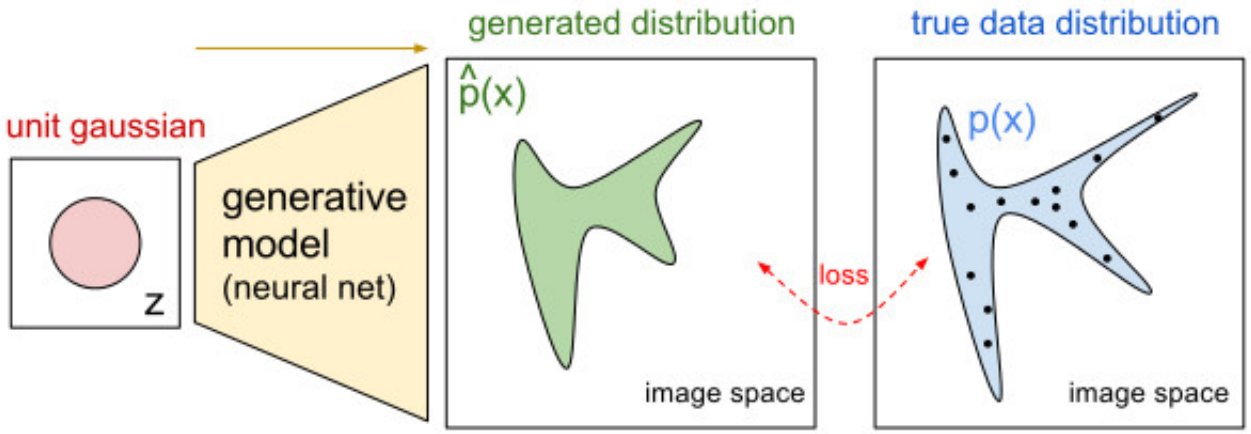
Generated sample $\sim \hat{P}(\boldsymbol{x})$

[L.Fei-Fei et al. CS231n]

Generative Models

Generative model using neural networks

- Given training images $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from $P(\boldsymbol{x})$
- Take samples from $\boldsymbol{z} \sim G(0, \sigma^2)$ and map them through neural networks (i.e. model distribution $\hat{P}(\boldsymbol{x})$)
- The model is trained so that $\hat{P}(\boldsymbol{x}) \sim P(\boldsymbol{x})$ as possible (i.e. parameters $\boldsymbol{\theta}$ are iteratively updated so that the green matches the blue distribution)



[openai tutorial]

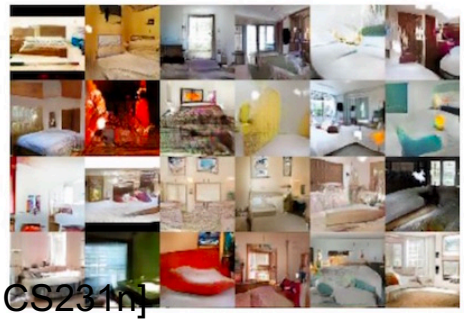
Generative Models

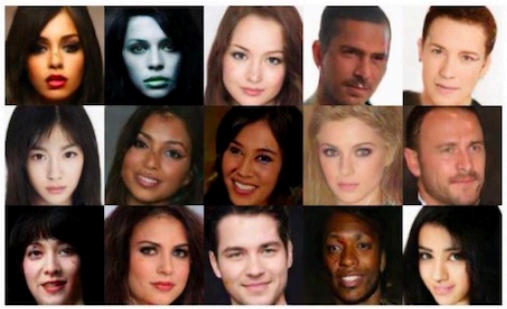
Two approaches

- Explicit density estimation (e.g. VAE): explicitly define and solve for $\hat{P}(\boldsymbol{x})$
- Implicit density estimation (e.g. GAN): learn model that can sample from $\hat{P}(\boldsymbol{x})$ w/o explicitly defining it

Applications

- Realistic samples for artwork, super-resolution, colorization, …
- Inference of latent representations
- Generation of time-series data for simulation and planning





[L.Fei-Fei et al. CS231n]

Generative Adversarial Networks

A flexible framework for generative modeling

- Formulated as a game between two players
- Straightforward to use neural networks as the players

Generator creates samples that are intended to come from the same distribution as the training data

- It tries to fool the discriminator

Discriminator examines the samples to determine whether they are real or fake

- A binary supervised classifier (real or fake for a given sample)
- It tries to detect faked data accurately as possible

I. Goodfellow et al. Generative Adversarial Nets. NIPS 2014.

Generative Adversarial Networks

Very well documented by the inventors (e.g. I. Goodfellow)

NIPS 2016 Tutorial: Generative Adversarial Networks

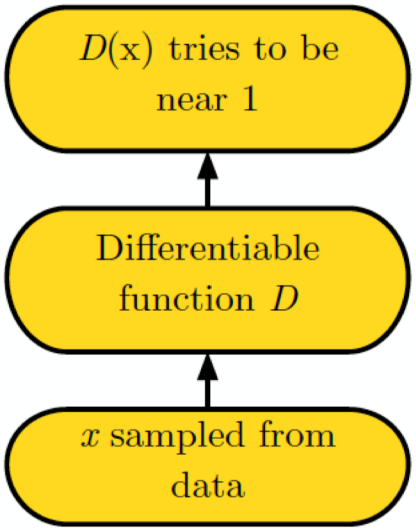
- Document: https://arxiv.org/abs/1701.00160
- Slides: http://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf

Codes of many GAN models available!

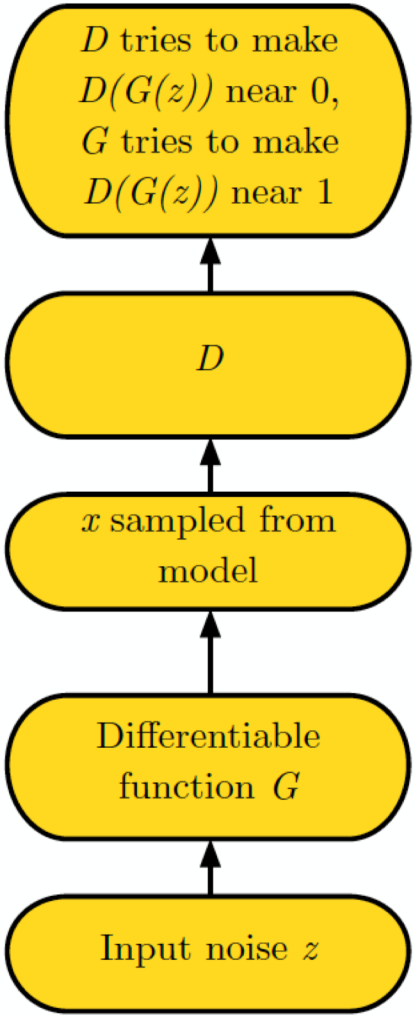
- Simply google the model name

Generative Adversarial Networks

Two scenarios for generator $G(\boldsymbol{z})$ and discriminator $D(\boldsymbol{x})$



1. $D(\boldsymbol{x})$ tries to assign a sample $\boldsymbol{x}$ from actual data to near 1



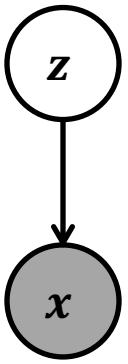
2. $D(\boldsymbol{x})$ tries to assign a sample generated by $G(\boldsymbol{z})$ to near 1

2. At the same time, $G(\boldsymbol{z})$ tries to generate a sample to be assigned to near 1 by $D(\boldsymbol{x})$

Generator and Discriminator

Generator $\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$ vs Discriminator $c = D(\boldsymbol{x}; \boldsymbol{\theta}^{(D)})$

- $\boldsymbol{z}$: (latent) input noise variable from a prior $P(\boldsymbol{z})$
- $\boldsymbol{x}$: observed variable, c: a scalar output
- $\boldsymbol{\theta}^{(G)}/\boldsymbol{\theta}^{(D)}$: parameters for generator/discriminator
- $J^{(G)}\left(\boldsymbol{\theta}^{(D)}; \boldsymbol{\theta}^{(G)}\right)$ / $J^{(D)}(\boldsymbol{\theta}^{(D)}; \boldsymbol{\theta}^{(G)})$: the cost function to be minimized for G/D
- Both should be differentiable w.r.t. their own input and parameters (will use gradient descent)



Cost Functions

Three popular options: (1) minimax, (2) non-saturating, (3) Maximum likelihood

1. Minimax game (i.e. zero-sum)

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log(1 - D\big(G(\boldsymbol{z})\big))$$

$$J^{(G)} = -J^{(D)}$$

cost for actual data $\boldsymbol{x}$ (desired $D(\boldsymbol{x}) = 1$)

cost for faked $G(\boldsymbol{z})$ (desired $D(\boldsymbol{x}) = 0$)

- The value of the game is $V(G, D) = -J^{(D)}$

Minimax – Solution to Discriminator

For any given G, what is the optimal discriminator D?

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

$$V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}} \log(1 - D\big(G(\boldsymbol{z})\big))$$

- Let the data distribution to be p_{data} and the distribution of samples $\boldsymbol{x}$ by $G(\boldsymbol{z})$ to be p_{mod}

$$V(G,D) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{x}\sim p_{mod}} \log(1 - D(\boldsymbol{x}))$$

$$= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log D(\boldsymbol{x})\, dx + \int_{\boldsymbol{x}} p_{mod}(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))\, dx$$

$$= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log D(\boldsymbol{x}) + p_{mod}(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))\, dx$$

Minimax – Solution to Discriminator

For any given G, what is the optimal discriminator D?

$$\max_D \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log D(\boldsymbol{x}) + p_{mod}(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))\, dx$$

- Use the fact that for any $(a, b) \in \mathbb{R}^2 \backslash \{0,0\}$, the function $y = a \log y + b \log(1 - y)$ has its maximum in $[0,1]$ at $a/(a + b)$

- The optimal $D_G^*(\boldsymbol{x})$ for any $p_{data}(\boldsymbol{x})$ and $p_{model}(\boldsymbol{x})$ is

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}$$

Minimax Solution

The training objective $C(G)$ is

$$C(G) = \max_D V(G, D) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}} \log D_G^*(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{x} \sim p_{mod}} \log(1 - D_G^*(\boldsymbol{x}))$$

$$= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{mod}(\boldsymbol{x})} + p_{model}(\boldsymbol{x}) \log \frac{p_{model}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{mod}(\boldsymbol{x})} dx$$

$$= -\log(4) + KL(p_{data}(\boldsymbol{x}) || \frac{p_{data}(\boldsymbol{x}) + p_{mod}(\boldsymbol{x})}{\mathbf{2}}) + KL(p_{mod}(\boldsymbol{x}) || \frac{p_{data}(\boldsymbol{x}) + p_{mod}(\boldsymbol{x})}{\mathbf{2}})$$

$$= -\log(4) + 2 \cdot JSD(p_{data}(\boldsymbol{x}) || p_{mod}(\boldsymbol{x}))$$

Minimax Solution

The global minimum of $C^*(G)$

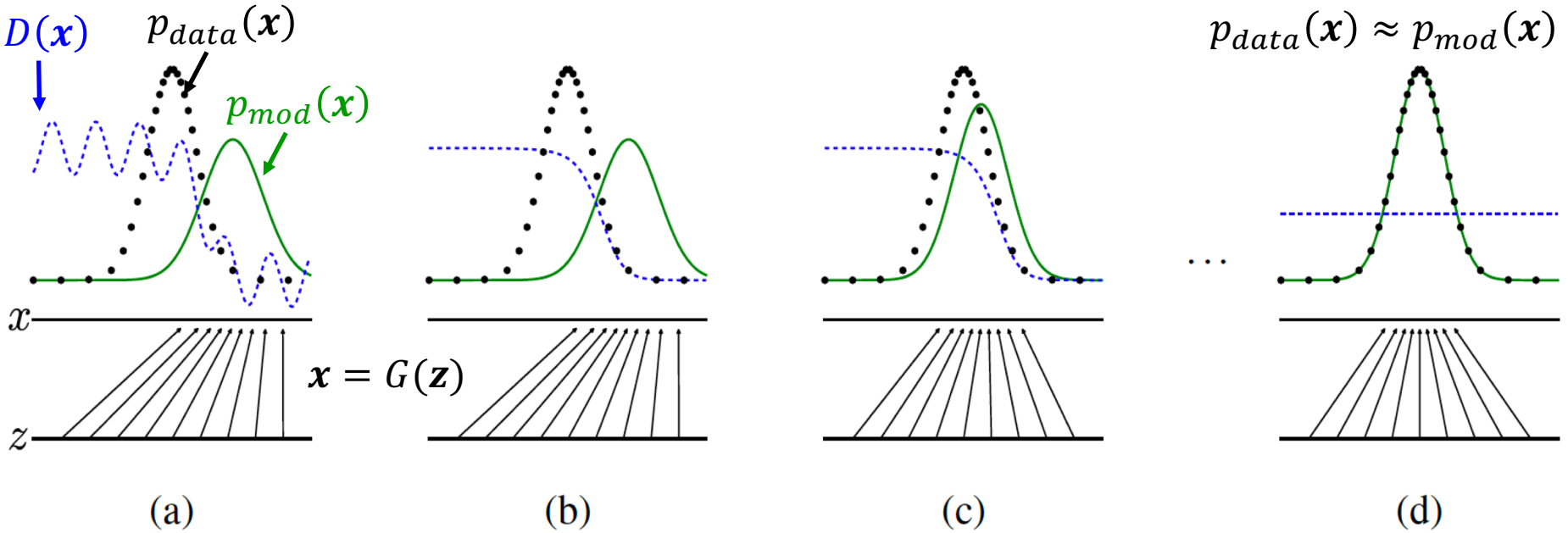
$$C(G) = \max_D V(G,D) = -\log(4) + 2 \cdot JSD(p_{data}(\boldsymbol{x}) || p_{mod}(\boldsymbol{x}))$$

$$C(G^*) = \max_G \max_D V(G,D) = -\log(4) \quad \text{at} \quad p_{data} = p_{mod}$$

- The solution $p_{data} = p_{mod}$: the generative model perfectly replicating the data generating process
- Resemble minimizing the Jensen-Shannon divergence between the data and the model distribution (i.e. $p_{data},\ p_{mod}$)
- For $p_{data} = p_{mod}$,

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})} = \frac{1}{2}$$

Learning Process of GAN



Starting converged

Fully converged

- The arrow shows how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ is done by generator
- (a) $D(\boldsymbol{x})$: partially accurate (e.g. $D(\boldsymbol{x})$ is high in the left)
- (b) $D(\boldsymbol{x})$ is converging to $D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}$
- (c) $p_{mod}(\boldsymbol{x})$ is moving to left guided by $D(\boldsymbol{x})$
- (d) $p_{data}(\boldsymbol{x}) \approx p_{mod}(\boldsymbol{x})$. $D^*(\boldsymbol{x}) \approx \frac{1}{2}$ cannot differentiate the two

Difficulty of GAN Training

Known to be very difficult to train

How to Train a GAN? Tips and tricks to make GANs work

- Up-to-date tips: https://github.com/soumith/ganhacks

README.md

How to Train a GAN? Tips and tricks to make GANs work

While research in Generative Adversarial Networks (GANs) continues to improve the fundamental stability of these models, we use a bunch of tricks to train them and make them stable day to day.

Here are a summary of some of the tricks.

Here's a link to the authors of this document

If you find a trick that is particularly useful in practice, please open a Pull Request to add it to the document. If we find it to be reasonable and verified, we will merge it in.

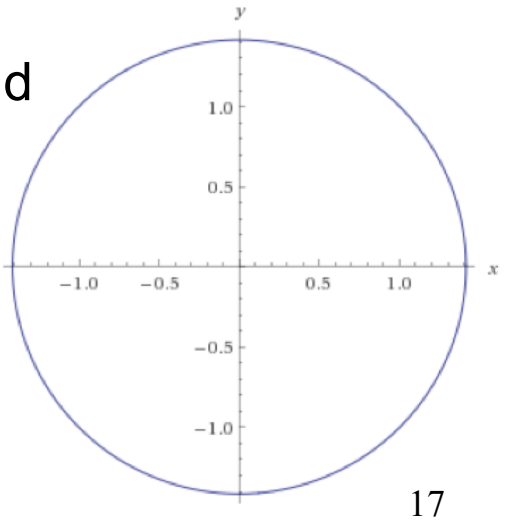
1. Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output

Difficulty of GAN Training

1. Non-convergence: optimization can oscillate between solutions

- It is a general problem with games
- Often the solutions of two-player games are saddle points
- e.g. Player 1 minimizes $V(x, y) = xy$ controling x
  Player 2 minimizes $V(x, y) = -xy$ controling y
- The equilibrium is $(0,0)$, but the gradient descent orbits forever around it (blue)
- If P1 infinitesimally decrease x, then P2 would increase y

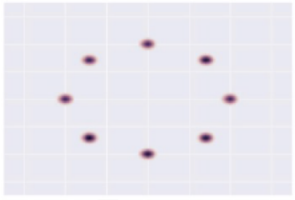


Difficulty of GAN Training

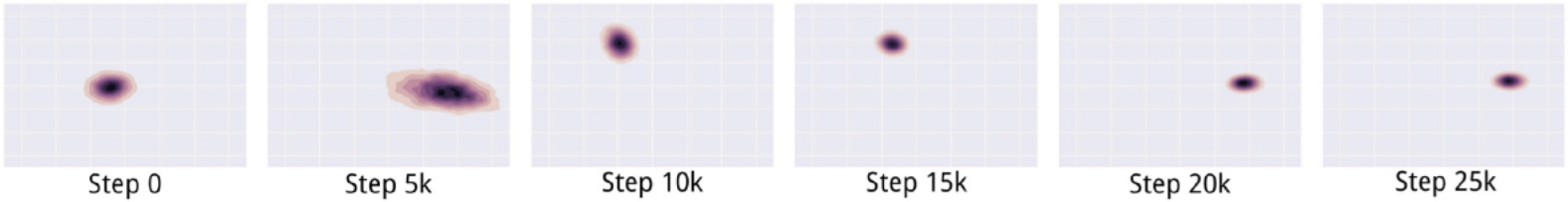
2. Mode collapse

- GANs often choose to generate from very few modes (fewer than actual # modes)
- Several different $\boldsymbol{z}$ values to the same output point
- Mode collapse causes low output diversity (e.g. more or less similar generated images)
- G only ever produces a single mode at a time, cycling between different modes as D learns to reject each one

Target distribution
p_{data}



Distributions that GAN learns over time

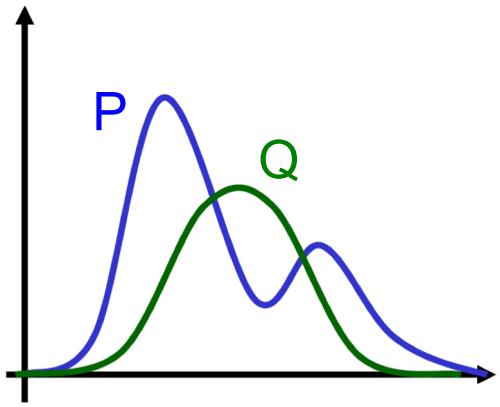


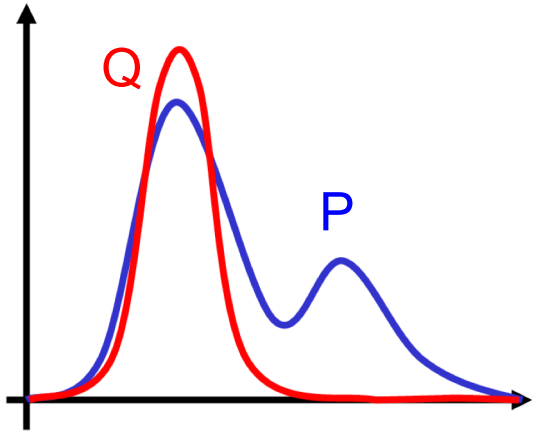
L. Metz et al. Unrolled generative adversarial networks. arXiv 2016

Difficulty of GAN Training

Why mode collapse happens ?

- Not fully understood yet, although the use of JSD is conjectured as a main cause
- Forward KL: prefer to place high probability everywhere that the data occurs
- Backward KL: prefer to place low probability wherever the data does not occur
- Backward KL can capture as many mode of data as the model can





Difficulty of GAN Training

3. Unstable dynamics: hard to keep G and D in balance
   - Often D overpower G; too accurate D can vanish the gradient for G
   - Do not limit the power of D, but to use regularizer where the gradient does not vanish

4. Quantitative evaluation of G is not easy
   - There is no clearly justified way to quantitatively score samples
   - It can be difficult to estimate the likelihood for GANs (that other generative models)

Deep Convolutional GAN

DCGAN: Use CNNs instead of MLPs

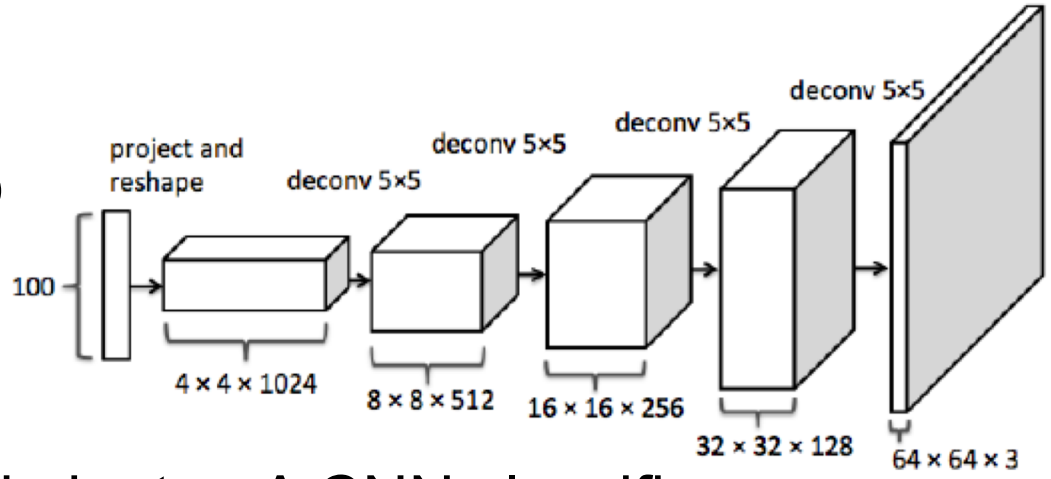
- Generate a realistic image that is different from training images
- Generator is optimized to fool D (more realistic images)
- Discriminator is optimized to not get fooled by G

A. Radford et al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. ICLR 2016.

DCGAN Model

Generator: A CNN mapping btw random codes and images

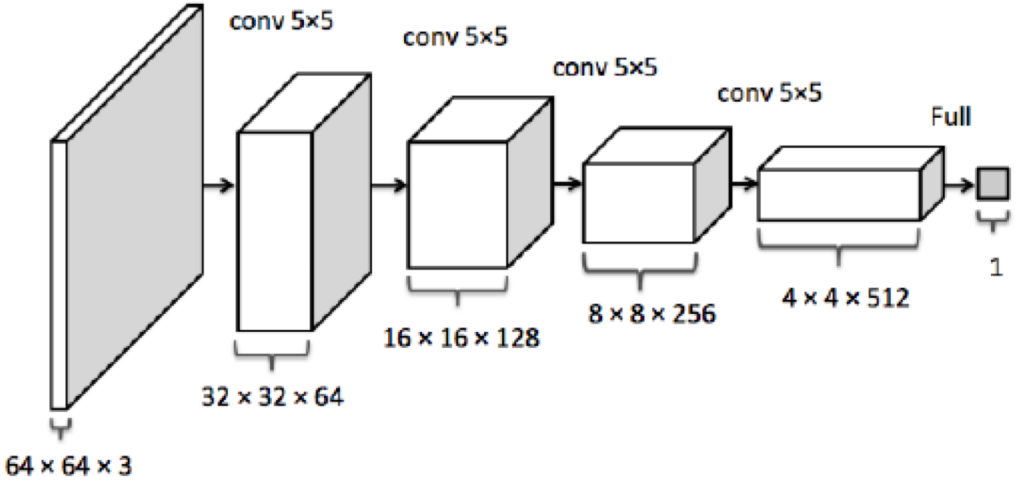
Input: 100-D random numbers



Output: 64x64 image

The discriminator: A CNN classifier

Input: 64x64 image



Output: real/fake score

DCGAN Model

Specific design of Generator (G) and discriminator (D)

- G : Replace any pooling layers with strided convolutions
- Use batchnorm in both G and D
- Uses Tanh for the output (and sigmod) in G
- Use LeakyReLU in D and ReLU in G
- Code: https://github.com/Newmu/dcgan_code

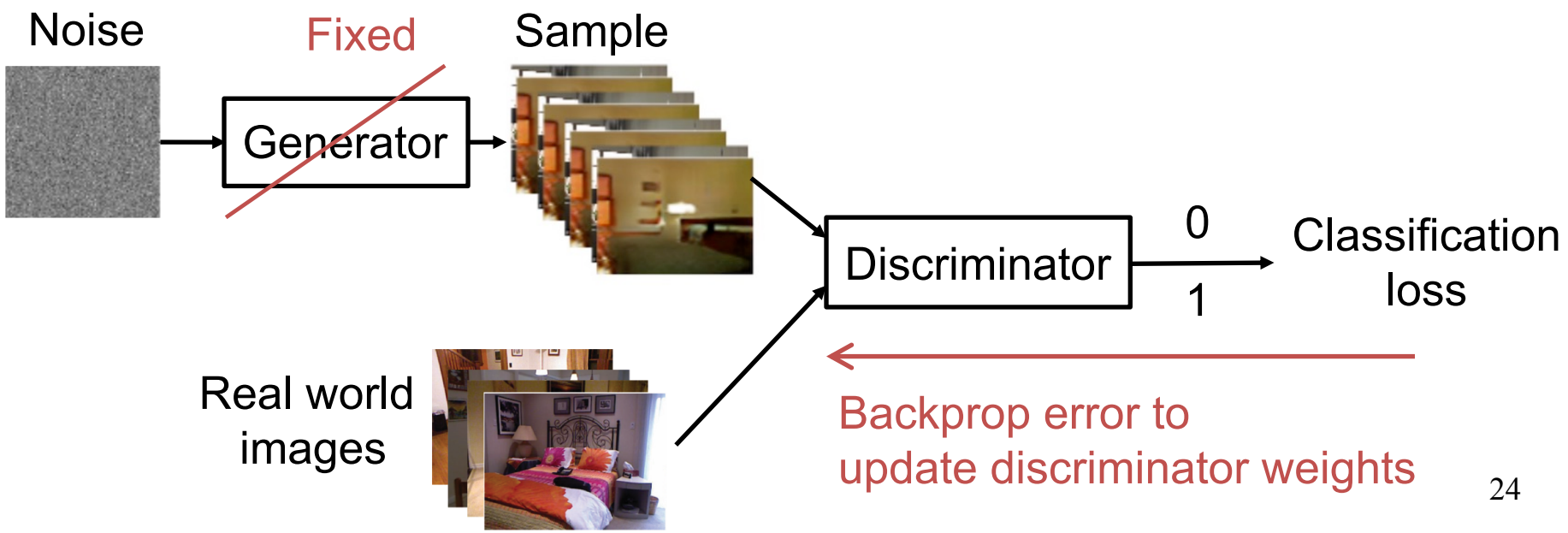
Training of DCGAN

Alternate between training discriminator and generator

- Note that both G and D should be differentiable

1. Training D

- Fix G's weights, draw a minibatch of samples from both real-world and generated images



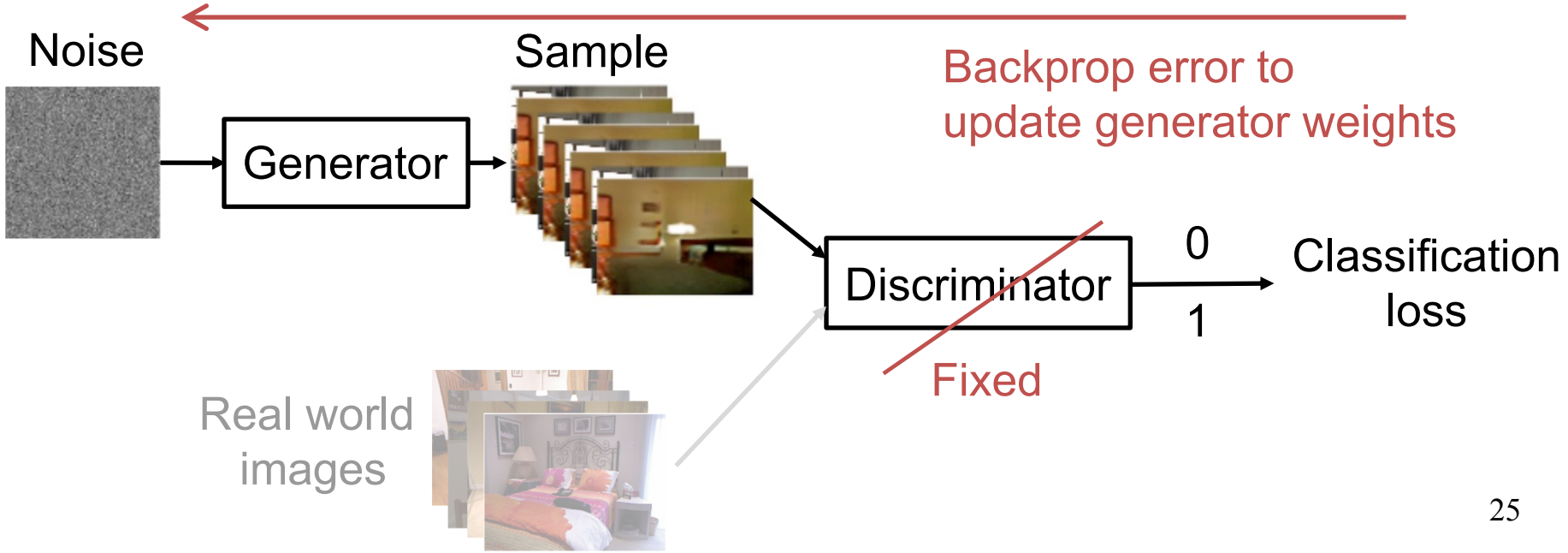
Training of DCGAN

Alternate between training discriminator and generator

- Note that both G and D should be differentiable

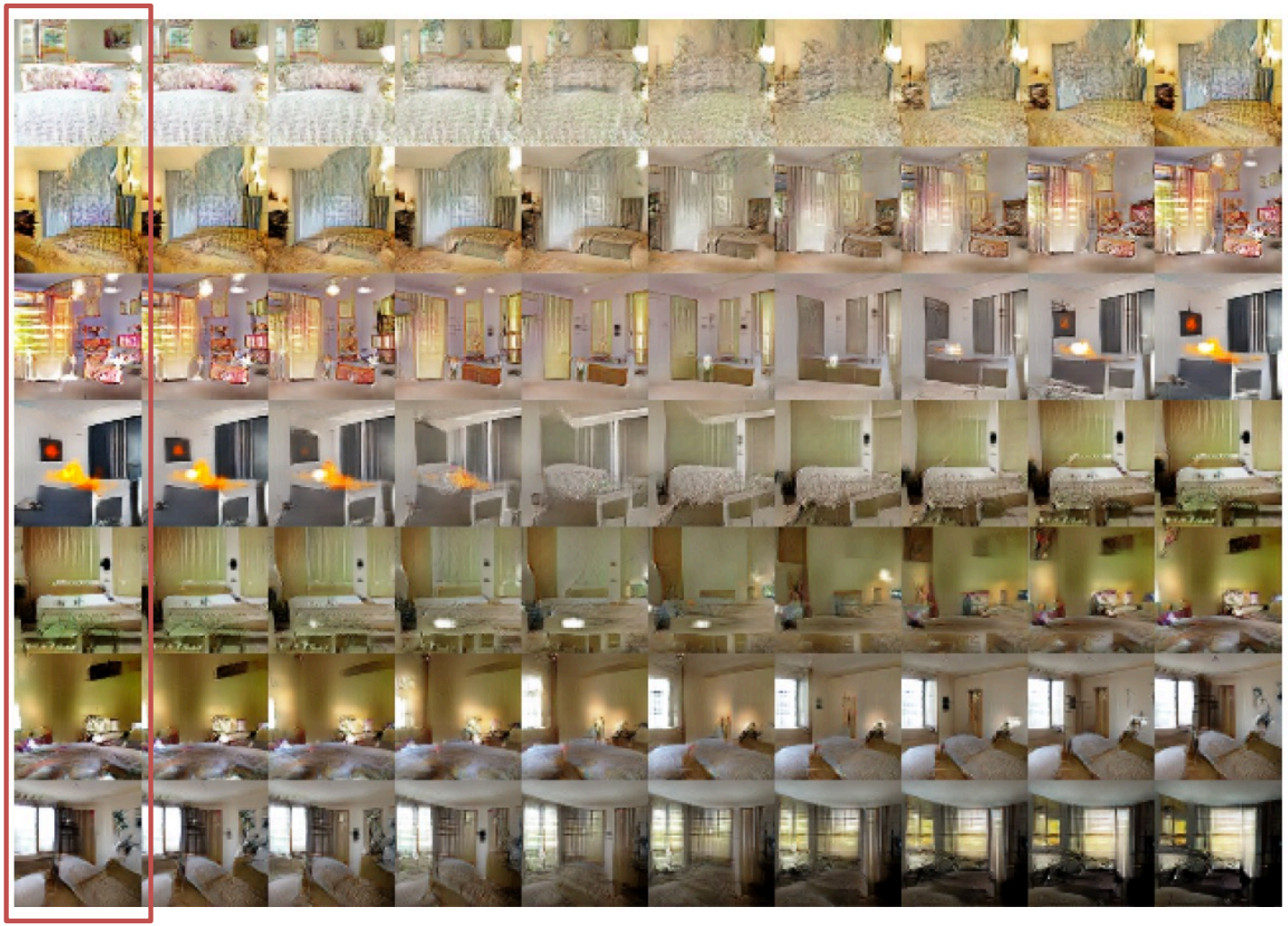
2. Training G

- Fix D's weights and a minibatch from G
- Backprop error through discriminator to update generator weights



DCGAN Results – Walking in the Latent Space

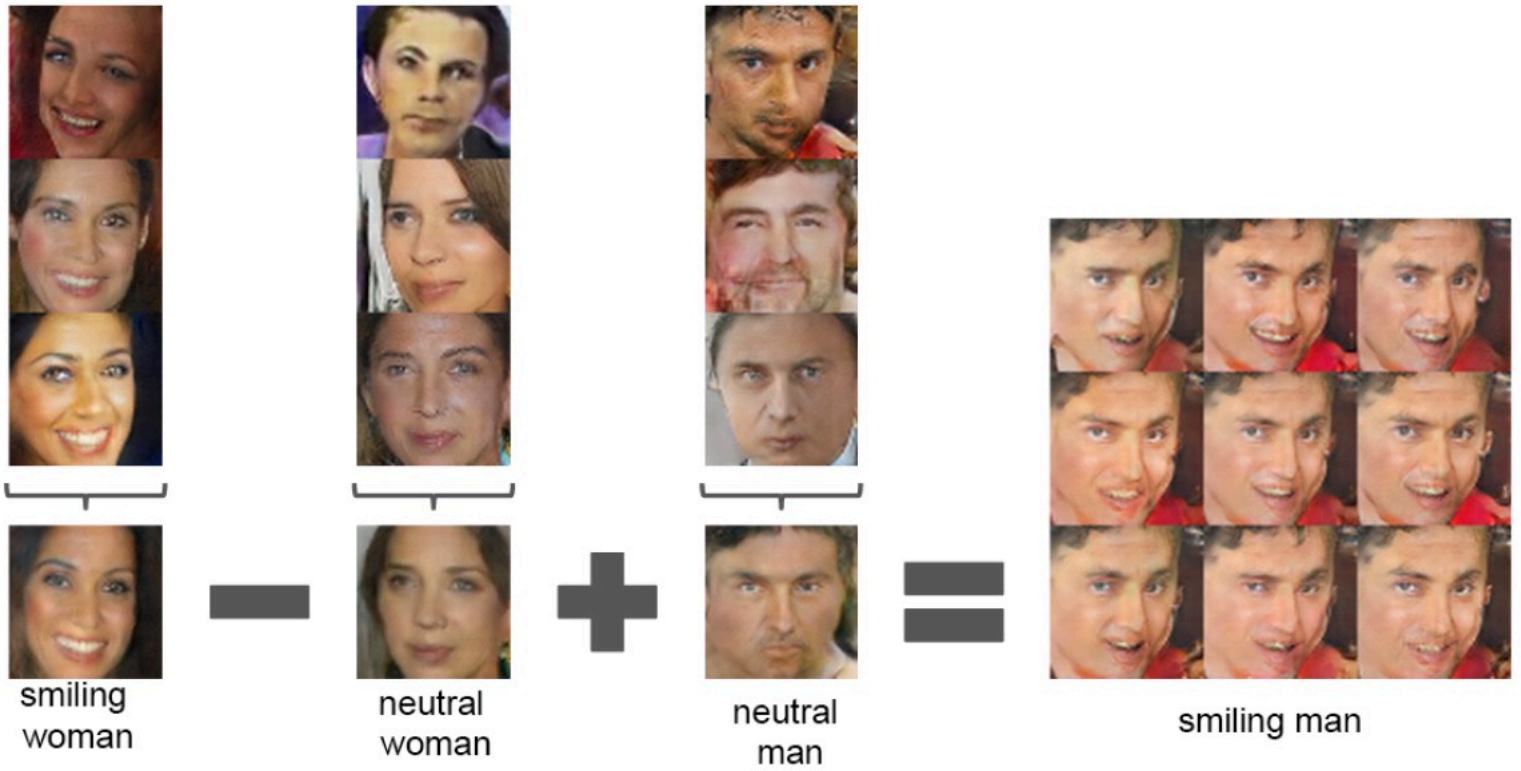
Pick 9 random points in z, and observe smooth transition between them



DCGAN Results – Vector Arithmetic

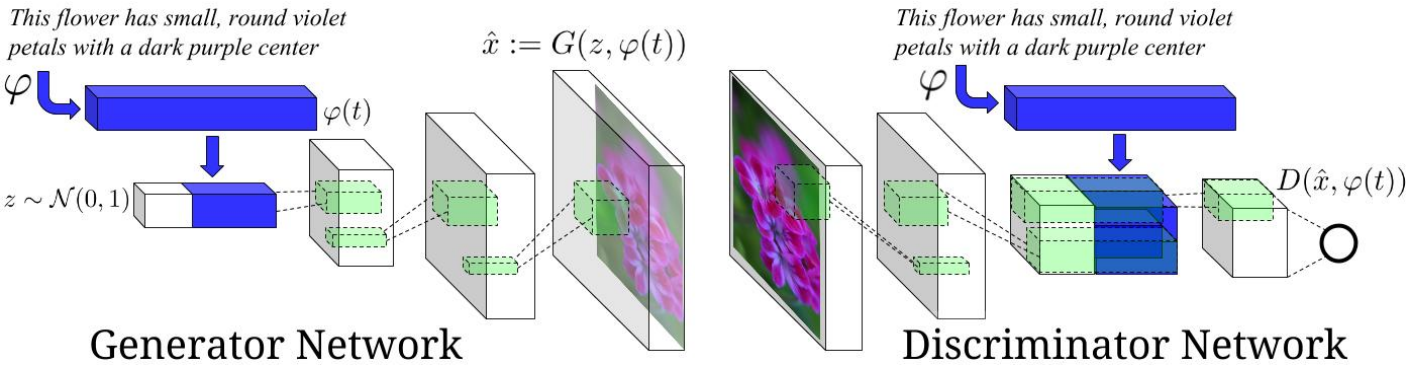
Use $\boldsymbol{z} = \boldsymbol{z}_{sw} - \boldsymbol{z}_{nw} + \boldsymbol{z}_{nm}$ to sample images, which turns out to be *smiling man*

- Use averaged $\boldsymbol{z}$ from three samples



Notable GAN Models – GAN-T2I

GAN-T2I (Text-to-Image Synthesis)



- Text is embedded using a FC-layer (128-D) and leaky ReLU
- Based on DC-GAN, the encoded text is concatenated into input random code in G, and convolutional layer in D
- Code: https://github.com/reedscot/icml2016

S. Reed et al. Generative Adversarial Text to Image Synthesis. ICML 2016.

Notable GAN Models – InfoGAN

InfoGAN (Interpretable GAN)

- In the GAN, the relation btw the random input code and the output image is entangled
- Interpretable (one dimension of the code for one aspect of images)
- Maximize the mutual information btw small subsets of the representation variables and the observation.
- Code: https://github.com/openai/InfoGAN

Azimuth (pose)

Elevation (pose)

X. Chen et al. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. ArXiv 2016.

Outline

- GANs
- Recent Advances in GANs

Image Synthesis

PGGAN [Brock et al., ICLR 2019]

StyleGAN [Karras et al., CVPR 2019]

[1] Brock et al., Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019
[2] Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

Image Synthesis with Generative Models

Generative models are trained to learn an image distribution p_{data}

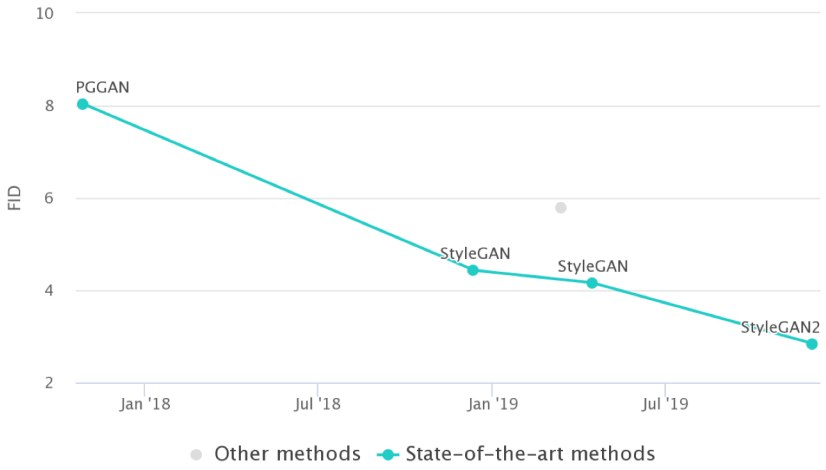
- VAE maximizes lower bound of $E_{p_{data}}[\log p_\theta(x)]$
- GANs minimize an estimate of some divergence metric between $p_{data}(x)$ and $p_\theta(x)$

Most of the modern image synthesis pipelines are based on GANs

- Samples from GANs are quite ‘crisp’ while samples from VAEs are usually ‘blurry’
- This sample quality gap is becoming smaller (e.g. VQ-VAE v2)
- But in this talk, we will mainly focus on GAN-based image synthesis

Status Quo of Image Synthesis on FFHQ

StyleGAN v2 [Karras et al., CVPR 2019]

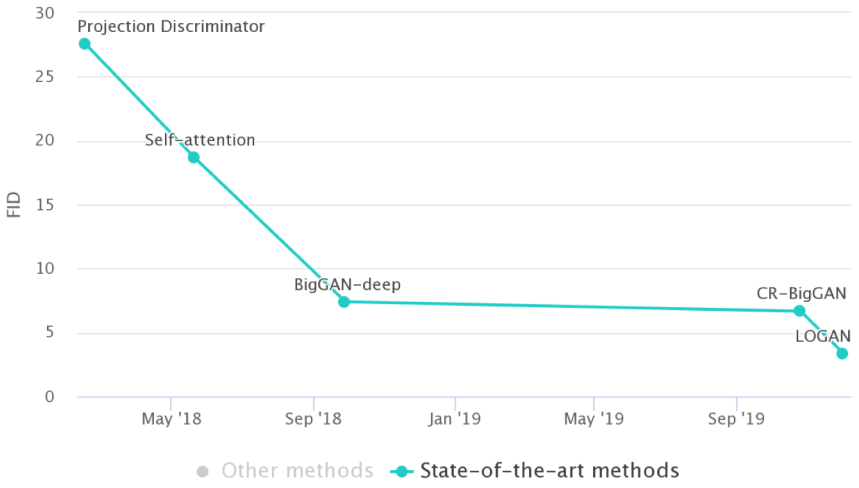


FID on FFHQ [2]

[1] Karras et al. Analyzing and Improving the Image Quality of StyleGAN, CVPR2019
[2] Flickr-Faces-HQ Dataset (FFHQ): https://paperswithcode.com/sota/image-generation-on-ffhq

Status Quo of Image Synthesis on ImageNet

LOGAN [Wu et al., 2019]



FID on ImageNet 128x128 [2]

[1] Wu et al. LOGAN: Latent Optimisation for Generative Adversarial Networks, ICLR 2019 (rejected)
[2] ImageNet 128x128 dataset: https://paperswithcode.com/sota/conditional-image-generation-on-imagenet

Challenges

GAN does a great job for producing high-fidelity samples

However, its training is notoriously difficult

- Training often collapses even on a simple toy dataset
- Empirically, this instability becomes larger with the size of model

To synthesize photorealistic images, we need

- A model that has enough capacity and right inductive bias to transform $p(z)$ to $p_{data}(x)$
- And a proper way to deal with training instability that comes with the scale

Progressive Growing GAN (PGGAN)

Goal: a new training method to produce images of unprecedented quality

Idea: propose to grow image resolution as training progresses

First photo-realistic samples (1024x1024) on CelebA-HQ

- vs. CIFAR-10 (32x32) and ImageNet (128x128)

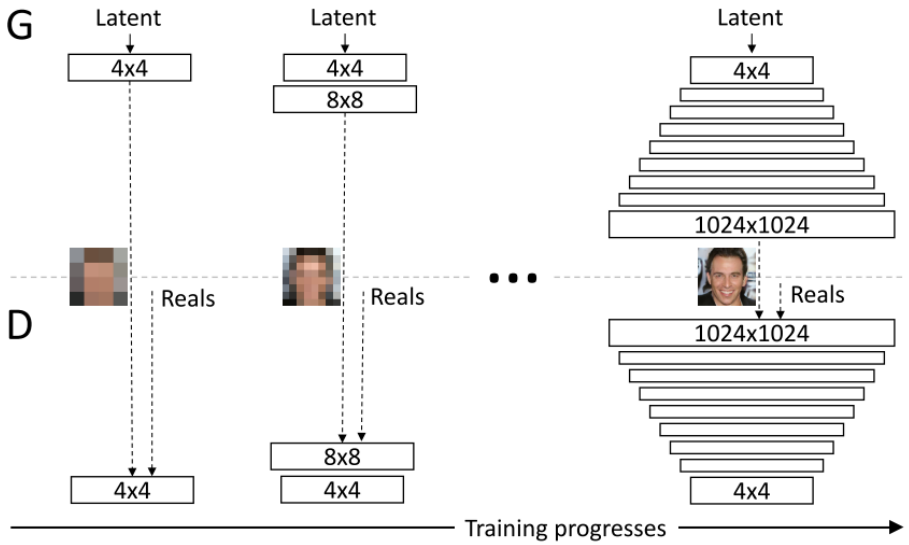
[1] Karras et al. (NVIDIA), Progressive Growing of GANs for Improved Quality, Stability, and Variation, ICLR 2018.

Progressive Growing GAN (PGGAN)

Issues of generation of high-resolution images

- It is much easier for D to capture unrealistic artifacts in high-resolution
- Necessitate using small minibatches due to memory constraint

Key idea: Grow resolution in a coarse-to-fine manner as training progresses



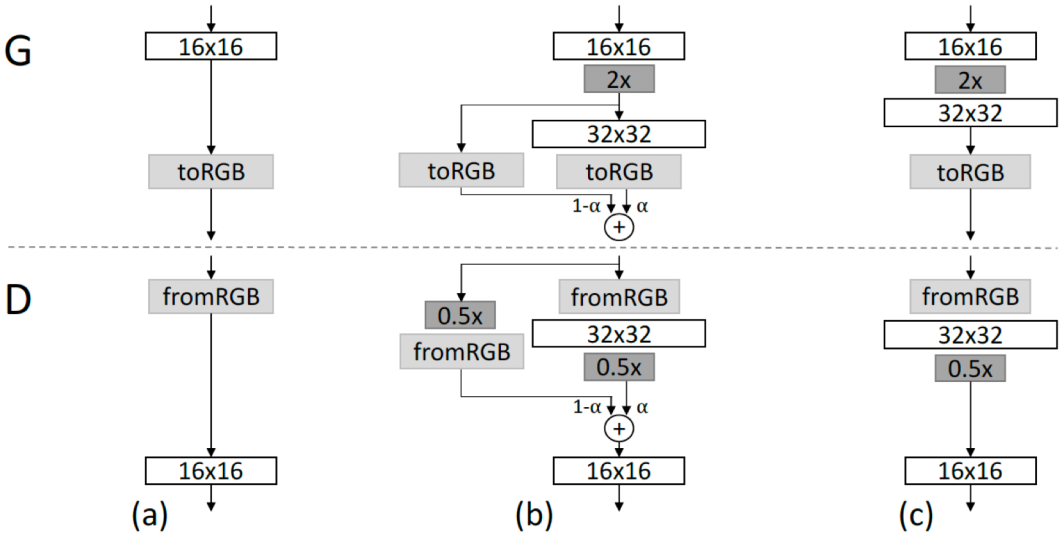
Progressive training

- Incrementally add layers to G and D
- All layers are trainable throughout the training
- Training is faster up to 2~6 times

PGGAN: How to Add a New Layer

Doubling the layer smoothly (a) → (b) → (c)

- The weight α increases linearly from 0 to 1
- The parameters in the 32x32 layer are gradually learned from 0 to 1



- 16x16/32x32: output layer (2 CONV + leaky ReLU)
- toRGB/fromRGB: the layer that projects feature vectors to RGB colors vice versa
- 2x: nearest neighbor interpolation
- 0.5x: average pooling

PGGAN: Other Heuristics for Stable Training

Neither learnable parameters nor hyperparameters

1. Minibatch standard deviation
   - Goal: Increase the variation of output samples
   - Add one additional constant feature map at the end of the discriminator
   - Compute a single number from the standard deviation of all features in a batch across spatial locations

PGGAN: Other Heuristics for Stable Training

2. Equalized learning rate

- Goal: Keep the weights in each network at a similar scale during the training
- Scale the weights at each layer with a constant (i.e. per-layer normalization constant)

3. Pixel-wise normalization

- Goal: Keep the magnitude of the feature map within a certain range
- Normalize the feature vector in each pixel to a unit length in the generator after each convolutional layer

PGGAN: Results (128x128) – Metrics

Training configuration	CELEBA						LSUN BEDROOM					
	Sliced Wasserstein distance $\times 10^3$					MS-SSIM	Sliced Wasserstein distance $\times 10^3$					MS-SSIM
	128	64	32	16	Avg		128	64	32	16	Avg	
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	**0.0587**
(b) + Progressive growing	4.62	**2.64**	3.78	6.06	4.28	**0.2838**	7.09	6.27	7.40	9.64	7.60	0.0615
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	**3.61**	3.32	**2.71**	6.44	4.02	0.0668
(g) + Pixelwise normalization	**4.06**	3.04	**2.02**	**5.13**	**3.56**	0.2845	3.89	**3.05**	3.24	**5.87**	**4.01**	0.0640
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636

- SWD (Sliced Wasserstein Distance) vs. MS-SSIM (multi-scale structural similarity)
- Existing metric (MS-SSIM) measures only the variation between outputs not similarity to training set
- Propose SWD: building Laplacian pyramid and randomly select patches from multiple layers and compute the Wasserstein distance between training and generated images
- Numbers: Levels of Laplacian pyramid

[1] Gulrajani et al., Improved Training of Wasserstein GANs, NIPS 2017.

PGGAN: Results (128x128) – Variants

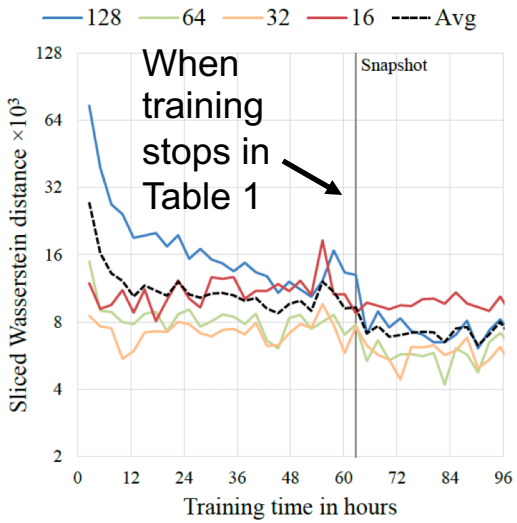
Training configuration	CELEBA						LSUN BEDROOM					
	Sliced Wasserstein distance $\times 10^3$					MS-SSIM	Sliced Wasserstein distance $\times 10^3$					MS-SSIM
	128	64	32	16	Avg		128	64	32	16	Avg	
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	**0.0587**
(b) + Progressive growing	4.62	**2.64**	3.78	6.06	4.28	**0.2838**	7.09	6.27	7.40	9.64	7.60	0.0615
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	**3.61**	3.32	**2.71**	6.44	4.02	0.0668
(g) + Pixelwise normalization	**4.06**	3.04	**2.02**	**5.13**	**3.56**	0.2845	3.89	**3.05**	3.24	**5.87**	**4.01**	0.0640
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636

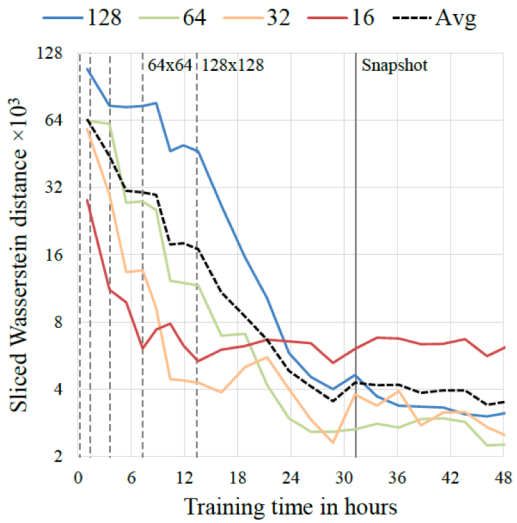
- (c) decrease minibatch size from 64 to 16
- (d) Adjust the hyperparameters as well as by removing batch/ layer normalization
- (e)(f)(g): the proposed heuristics
- Non-converged: terminating the training once the D has been shown a total of 10M real images

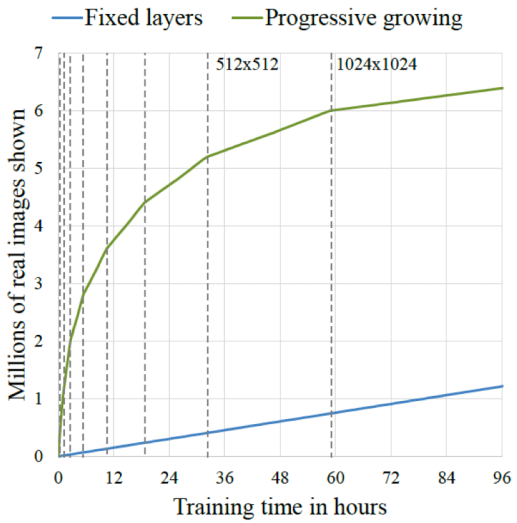
PGGAN: Speed and Convergence

Wasserstein GANs

Progressive growing enabled







On a single GPU (NVIDIA Tesla P100)

Raw training speed with 1024x1024

(dashed: double the resolution of G and D)

PGGAN: Summary

It's much easier for D to capture unrealistic artifacts in high-resolution

This leads to overwhelming D, which contributes unstable training

Progressive growing both stabilizes and accelerates the training

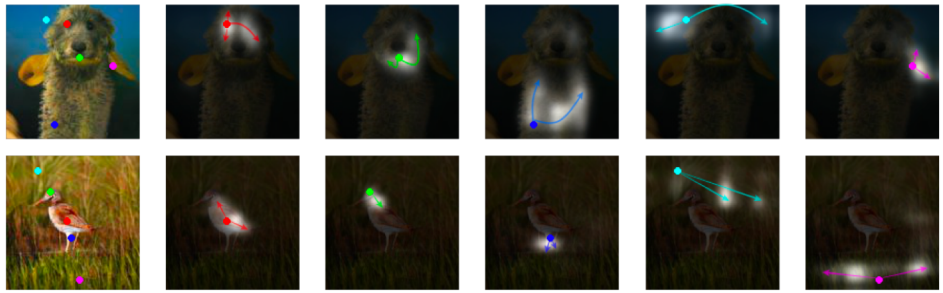
Self-Attention GAN (SA-GAN)

Introduce self-attention mechanism into GANs

- Improve long-range structure modelling (using the non-local model of [Wang et al. 2018])
- CNNs and RNNs are not efficient for modeling long-range dependencies in images

Two techniques for stabilization of GAN training

- Spectral Normalization [Miyato et al. 2018] and Two timescale update rule (TTUR) [Heusel et al. 2017]



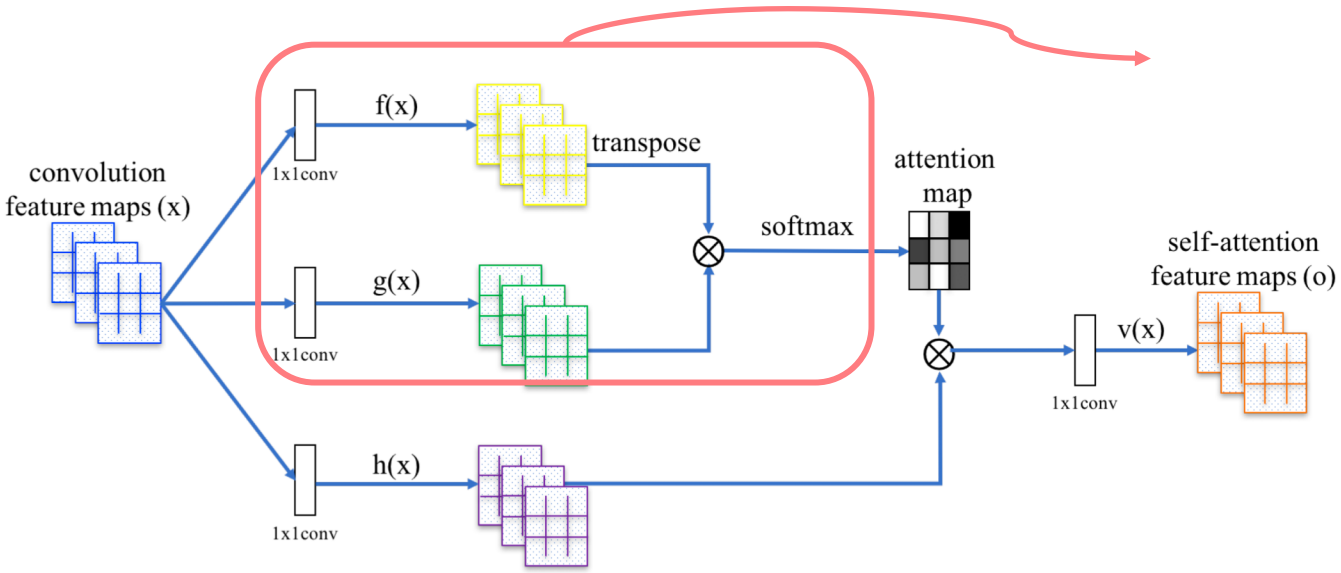
Attention maps for each of five query points

[1] Zhang et al. (Google), Self-Attention Generative Adversarial Networks, ICML 2019.
[2] Wang et al., Non-local Neural Networks, CVPR2019.
[3] Miyato et al., Spectral Normalization for Generative Adversarial Networks, ICLR 2018.
[4] Heusel et al. GANs trained by a two time-scale update rule converge to a local Nash equilibrium, NIPS 2017.

Self-Attention GAN (SA-GAN)

Generate self-attention map $\beta_{j,i}$

- Represents the extent to which the model attends to the i^{th} location when synthesizing the j^{th} region
- Compute j^{th} feature as a linear combination of transformed feature $h(\boldsymbol{x_i})$ based on self-attention map $\beta_{i,j}$



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

$$s_{ij} = \boldsymbol{f}(\boldsymbol{x}_i)^T \boldsymbol{g}(\boldsymbol{x}_j)$$

$$\boldsymbol{f}(\boldsymbol{x}_i) = \boldsymbol{W}_f \boldsymbol{x}_i$$
$$\boldsymbol{g}(\boldsymbol{x}_j) = \boldsymbol{W}_g \boldsymbol{x}_j$$

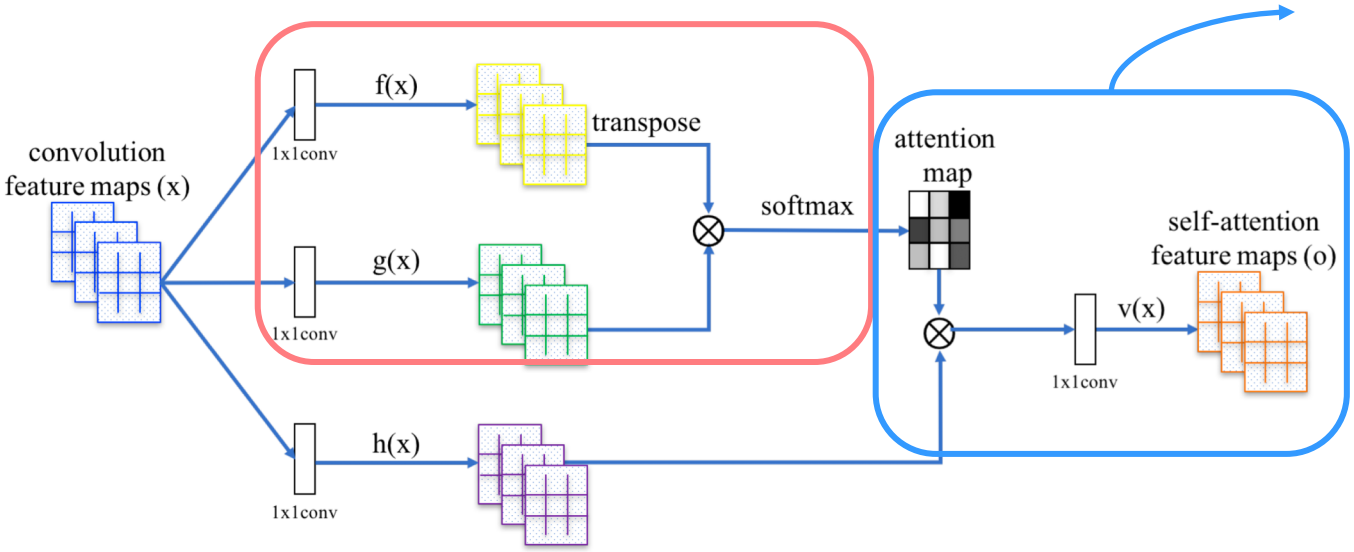
$$\boldsymbol{x} \in \mathbb{R}^{C \times N}$$

C: # of channels
N: # of feature locations

Self-Attention GAN (SA-GAN)

Generate self-attention map $\beta_{j,i}$

- Represents the extent to which the model attends to the i^{th} location when synthesizing the j^{th} region
- Compute j^{th} feature as a linear combination of transformed feature $h(\boldsymbol{x_i})$ based on self-attention map $\beta_{i,j}$



$$\boldsymbol{o}_j = \boldsymbol{v}(\sum_{i=1}^{N} \beta_{j,i}\boldsymbol{h}(\boldsymbol{x}_i))$$

$$\boldsymbol{h}(\boldsymbol{x}_i) = \boldsymbol{W}_h\boldsymbol{x}_i$$
$$\boldsymbol{v}(\boldsymbol{x}_i) = \boldsymbol{W}_v\boldsymbol{x}_i$$

Finally,

$$\boldsymbol{y}_i = \gamma\boldsymbol{o}_i + \boldsymbol{x}_i$$

SA-GAN: Results on ImageNet (128x128)

Model	no attention	SAGAN				Residual			
		$feat_8$	$feat_{16}$	$feat_{32}$	$feat_{64}$	$feat_8$	$feat_{16}$	$feat_{32}$	$feat_{64}$
FID	22.96	22.98	22.14	**18.28**	18.65	42.13	22.40	27.33	28.82
IS	42.87	43.15	45.94	51.43	**52.52**	23.17	44.49	38.50	38.96

Self-attention at the middle-to-high level feature maps (e.g. $feat_{32/64}$) achieves better performance

- $feat_k$: Adding self-attention to k x k feature maps
- Small feature map ~ convolution

Self-attention shows better performance than residual blocks (with the same # of parameters)

SA-GAN: Stabilization of GAN training

Spectral Normalization [Miyato et al. 2018]

- This regularization of the discriminator often slows down the GANs' training process
- Require multiple discriminator update steps (e.g. 5) per generator update step

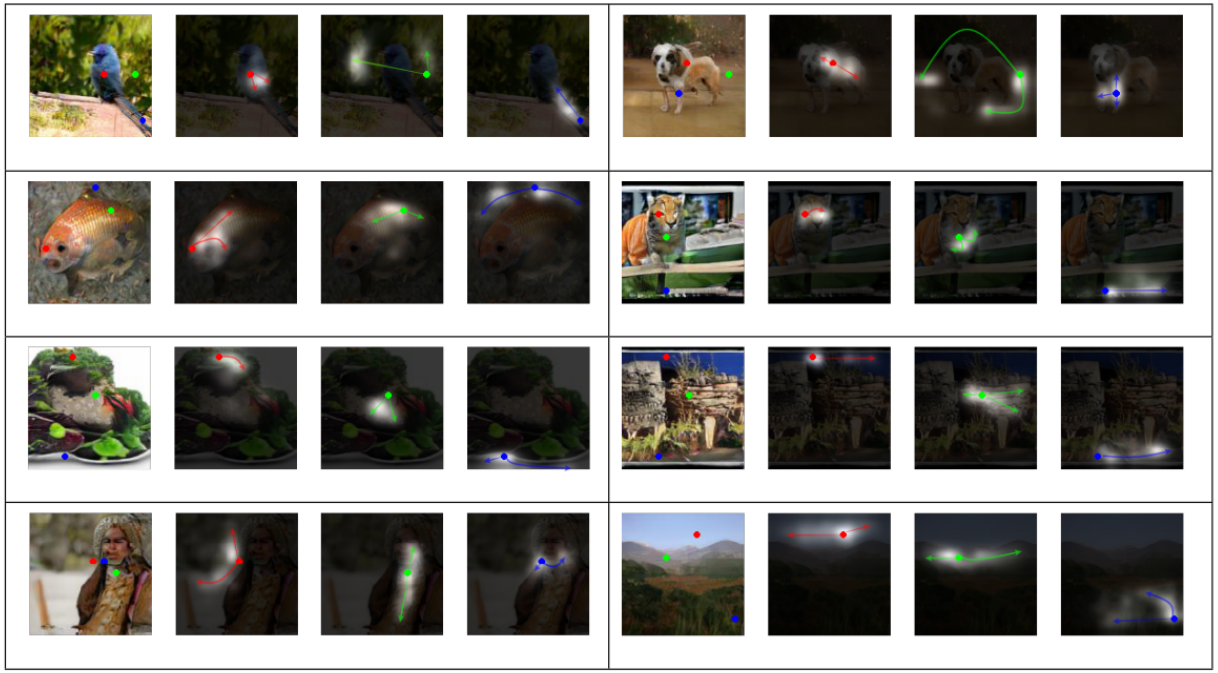
Two timescale update rule (TTUR) [Heusel et al. 2017]

- Compensate for the problem of slow learning in a regularized discriminator
- Use separate learning rates for generator and discriminator

SA-GAN: Visualization of Attention Maps

SA module to the last G layer (i.e. closest to output pixels)

Query points Self-attention maps



Color coded arrows summarizing the most-attended regions

The network learns to allocate attention according to similarity of color and texture, rather than just spatial adjacency

SA-GAN: Summary

CNN-based GANs excel at generating high-fidelity textures

But they often fail to capture long-range geometric/structural patterns

Self-attention improves long-range dependency modelling

BigGAN

Successfully train a large-scale GAN with some tricks

- A simple sampling technique (truncation trick) allows fine-grained control of the trade-off between sample variety and fidelity

GANs benefit dramatically from scaling

- Train models with 2-4x as many parameters and 8x batch size compared to prior art
- Huge improvement on class-conditional ImageNet compared to StyleGAN: IS 52.52 → 166.3 (x 3.19) and FID 18.65 → 9.6 (x 1.94)

[1] Brock et al. (DeepMind), Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019.

BigGAN: Scaling Up GANs

Use SA-GAN architecture as a baseline

- Both increased model / batch size improve FID and IS

Using skip connections from the latent z to multiple layers of G

Stable to 10^6 iterations, or it collapses at the given iteration

Batch size

units per layer

Shared embedding

Orthogonal Regularization

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(±1.18)
1024	64	81.5	✗	✗	✗	1000	14.88	63.03(±1.42)
2048	64	81.5	✗	✗	✗	732	12.39	76.85(±3.83)
2048	96	173.5	✗	✗	✗	295(±18)	9.54(±0.62)	92.98(±4.27)
2048	96	160.6	✓	✗	✗	185(±11)	9.18(±0.13)	94.94(±1.32)
2048	96	158.3	✓	✓	✗	152(±7)	8.73(±0.45)	98.76(±2.84)
2048	96	158.3	✓	✓	✓	165(±13)	8.51(±0.32)	99.31(±2.10)
2048	64	71.3	✓	✓	✓	371(±7)	10.48(±0.10)	86.90(±0.61)

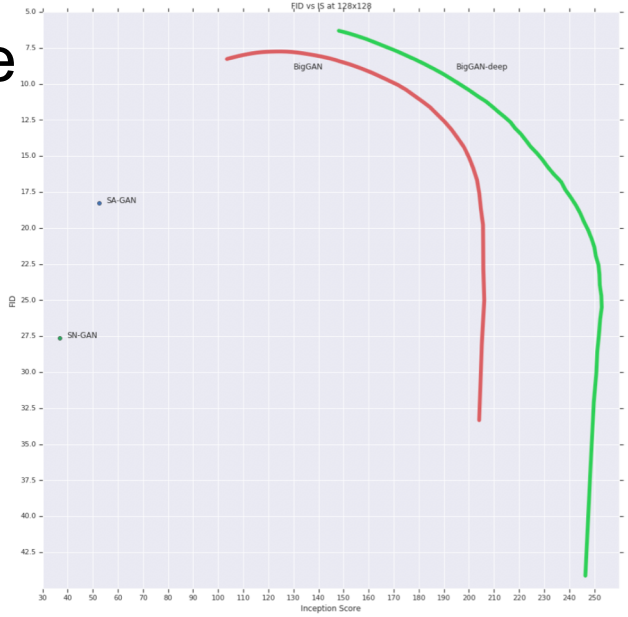
BigGAN: Scaling Up GANs

Truncating elements of z during the test time gives a huge improvement on IS

- The model is trained with $z \sim N(0, I)$ and samples z at test time so that the value that falls outside a range is resampled to be inside that range

Trade-off between FID and IS can be adjusted with the truncation level

- Frechet Inception Distance (FID) penalizes lack of diversity but Inception Score (IS) does not
- Reducing the truncation threshold diminishes variety
- A smaller threshold leads to a higher IS but a lower FID



BigGAN-deep:
4x layers

BigGAN: Results

ImageNet

		w/o Truncation	At the best FID setting	With validation data	At the best IS setting
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

JFT-300 (Google Internal dataset with 18K categories)

Ch.	Param (M)	Shared	Skip-z	Ortho.	FID	IS	(min FID) / IS	FID / (max IS)
64	317.1	✗	✗	✗	48.38	23.27	48.6/23.1	49.1/23.9
64	99.4	✓	✓	✓	23.48	24.78	22.4/21.0	60.9/35.8
96	207.9	✓	✓	✓	18.84	27.86	17.1/23.3	51.6/38.1
128	355.7	✓	✓	✓	13.75	30.61	13.0/28.0	46.2/47.8

BigGAN: Summary

ResNet architecture is powerful enough to achieve SOTA on ImageNet

Both increased model / batch size improves FID and IS

Truncation trick gives a huge improvement on IS

StyleGAN

Proposes a new generator architecture inspired from style transfer literature

- Use the AdaIN (Adaptive Instance Normalization) operations of [2]

Achieves SOTA on CelebA-HQ and FFHQ (Flickr-Faces-HQ, proposed)

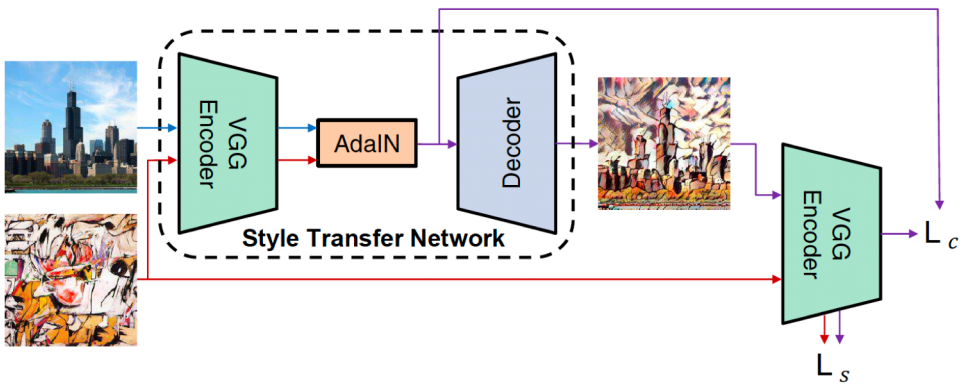
[1] Karras et al. (NVIDIA), A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019.
[2] Huang & Belongie, Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, CVPR 2017.

StyleGAN: Review of AdaIN

Adaptive Instance Normalization (AdaIN)

- Receive a content input x and a style input y
- Affine transformation (with no learnable parameter) to align the channel-wise mean and variance of x to match those of y

$$\text{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$



Encoder: first few layers of a fixed VGG-19

Decoder: mirrors the encoder, with all pooling layers replaced by nearest up-sampling to reduce checkerboard effects

L_c: content loss, L_s: style loss

[1] Huang & Belongie, Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, CVPR 2017.

StyleGAN: A New Generator Architecture

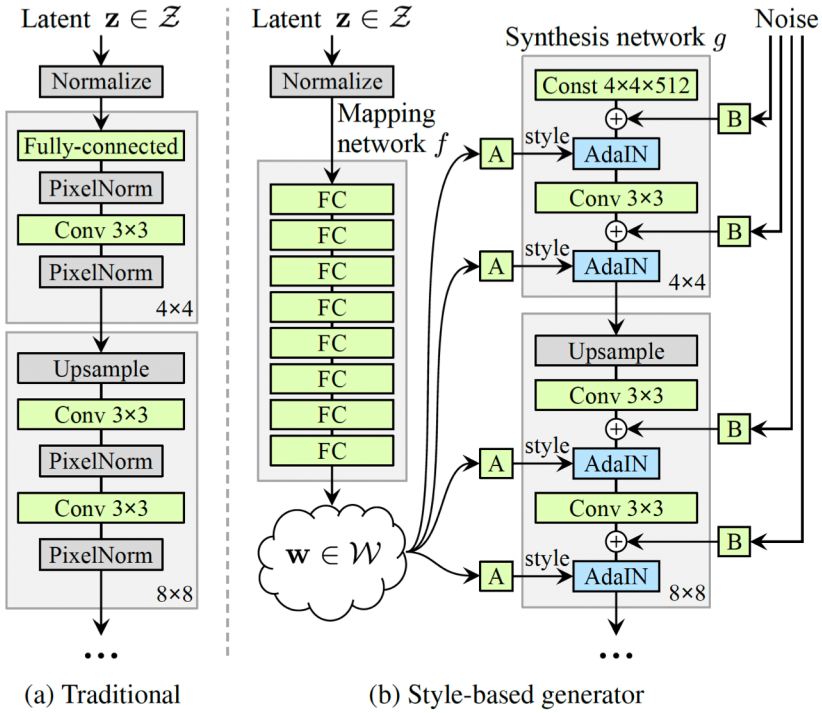
An 8 FC-layered Mapping network

- Transforms a latent code $\boldsymbol{z}$ to the embedded latent code $\boldsymbol{w}$
- Traditionally, the latent code is provided to the generator through an input layer

For each layer

- Latent $\boldsymbol{w}$ is transformed to layer-specific styles $(\boldsymbol{y}_s, \boldsymbol{y}_b)$ via an affine transformation [A]
- Then $(\boldsymbol{y}_s, \boldsymbol{y}_b)$ control the style of feature maps $\boldsymbol{x}_i$ with AdaIN operation

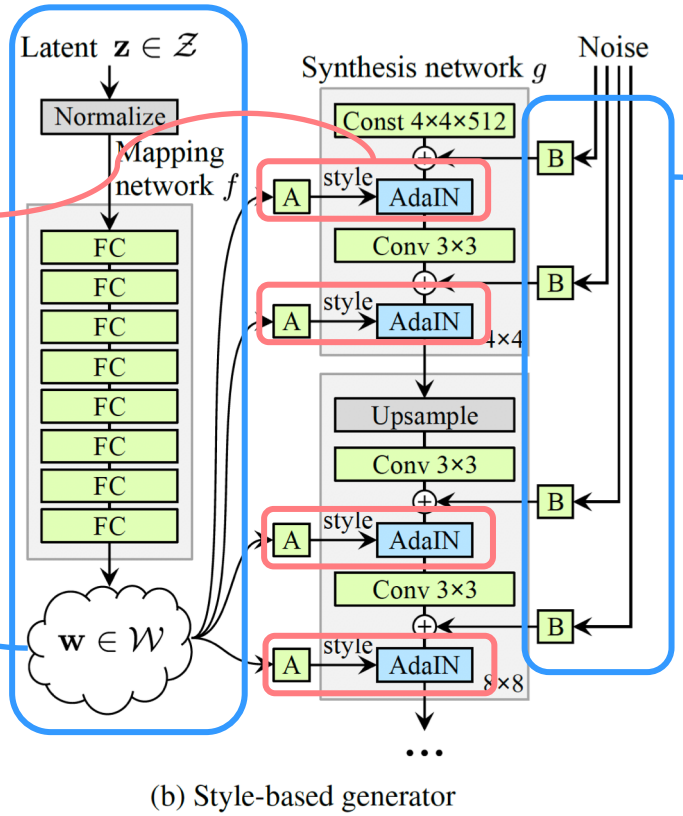
$$\mathrm{AdaIN}(\boldsymbol{x}_i, \boldsymbol{y}) = \boldsymbol{y}_{s,i} \left(\frac{\boldsymbol{x}_i - \mu(\boldsymbol{x}_i)}{\sigma(\boldsymbol{x}_i)} \right) + \boldsymbol{y}_{b,i}$$



StyleGAN: A New Generator Architecture

Control overall style of each feature maps with y_s, y_b

Sample w from complex distribution



(b) Style-based generator

Layer-wise noise for high-frequency details (e.g. uncorrelated Gaussian)

Added after each CONV layer

A total of 26.2M trainable parameters (vs. 23.1M in the traditional G)

StyleGAN: Ablation Study

FID scores for various generator designs

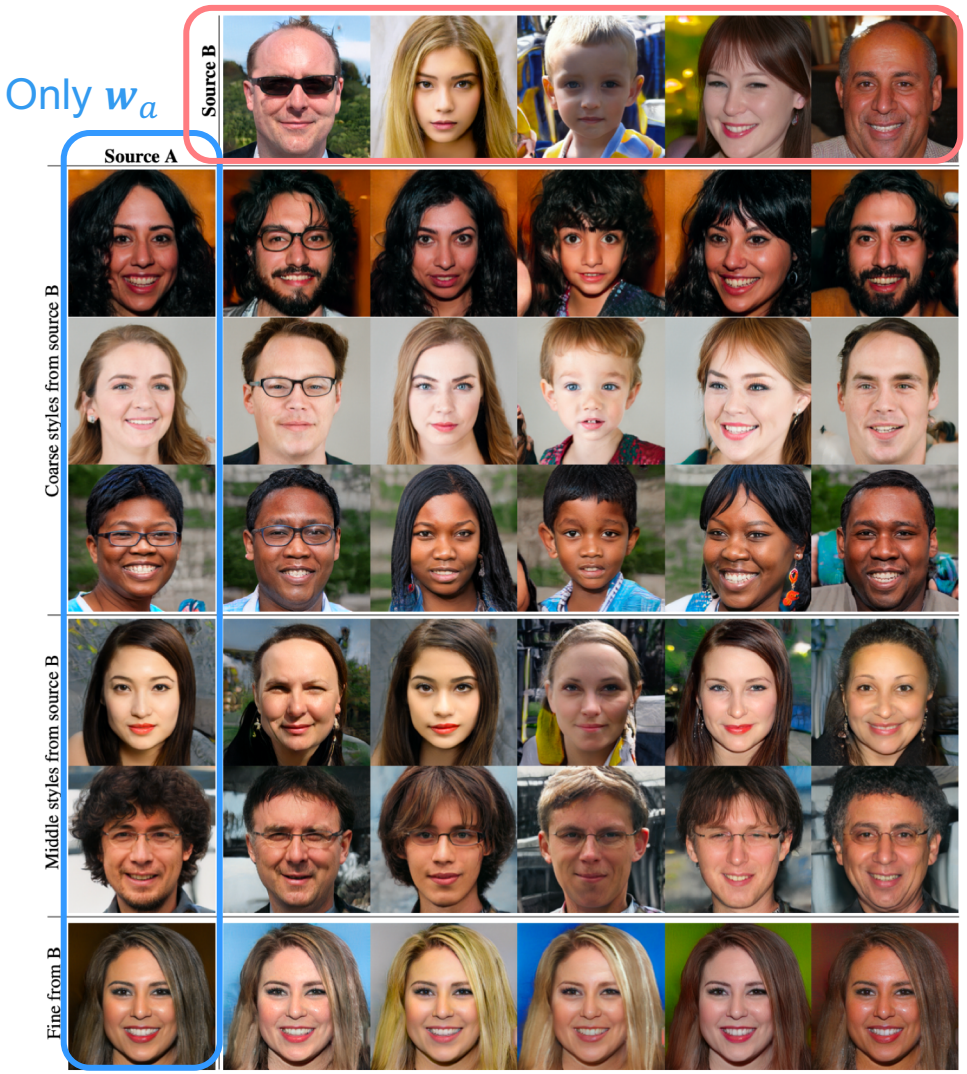
- (a) Baseline and (b) its improved version (e.g. longer training, bilinear up/down sampling)
- (c) Adding AdaIN operations
- (d) Simplifying architecture by removing the traditional input layer but starting from $\boldsymbol{w}$
- (e) Introducing the noise input
- (f) Mixing regularization (see later)

	Method	CelebA-HQ	FFHQ
A	Baseline Progressive GAN [30]	7.79	8.04
B	+ Tuning (incl. bilinear up/down)	6.11	5.25
C	+ Add mapping and styles	5.34	4.85
D	+ Remove traditional input	5.07	4.88
E	+ Add noise inputs	**5.06**	4.42
F	+ Mixing regularization	5.17	**4.40**

StyleGAN: Ablation Study

Generate an image using two random latent codes

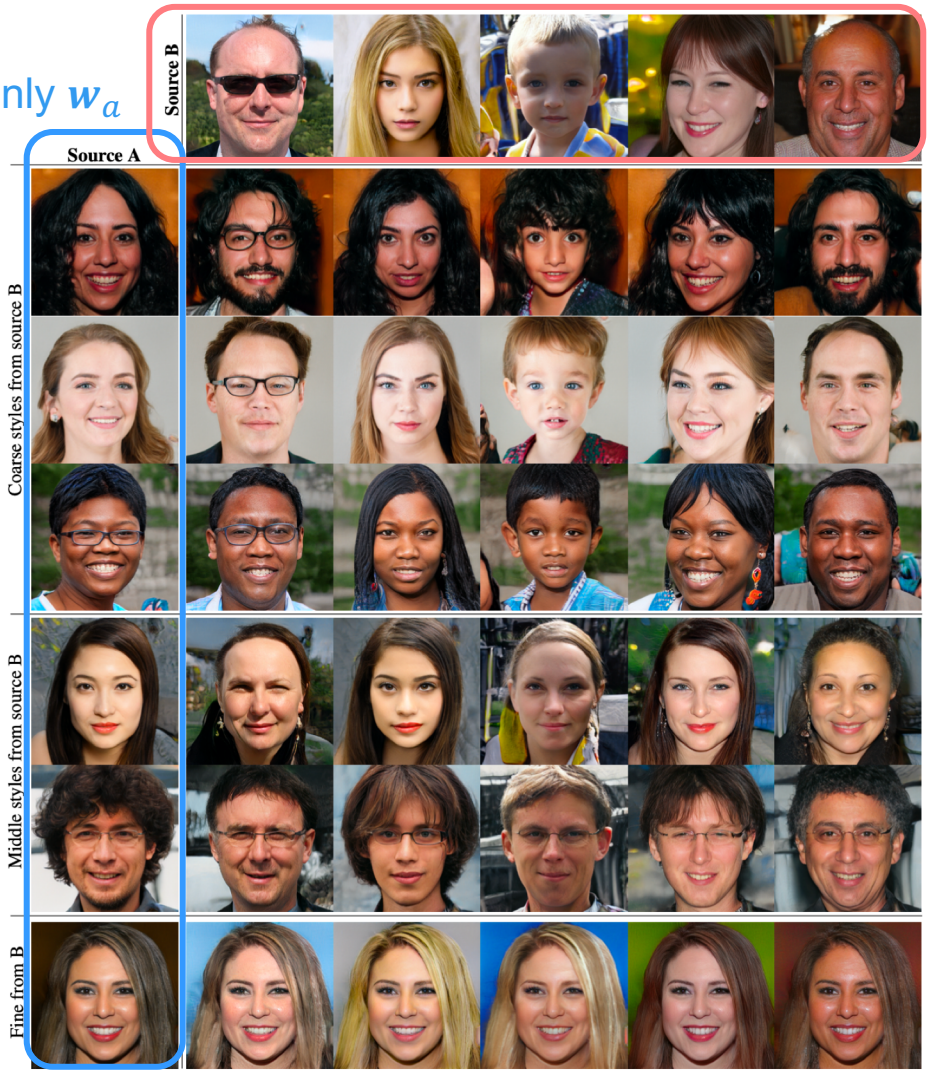
- Obtain $\boldsymbol{w}_a$ and $\boldsymbol{w}_b$ from two latent codes
- A split point in the synthesis network is chosen and $\boldsymbol{w}_a$ is used for AdaIN operations before the point while $\boldsymbol{w}_b$ is after the point



StyleGAN: Ablation Study

In the examples, $\boldsymbol{w_b}$ is used ...

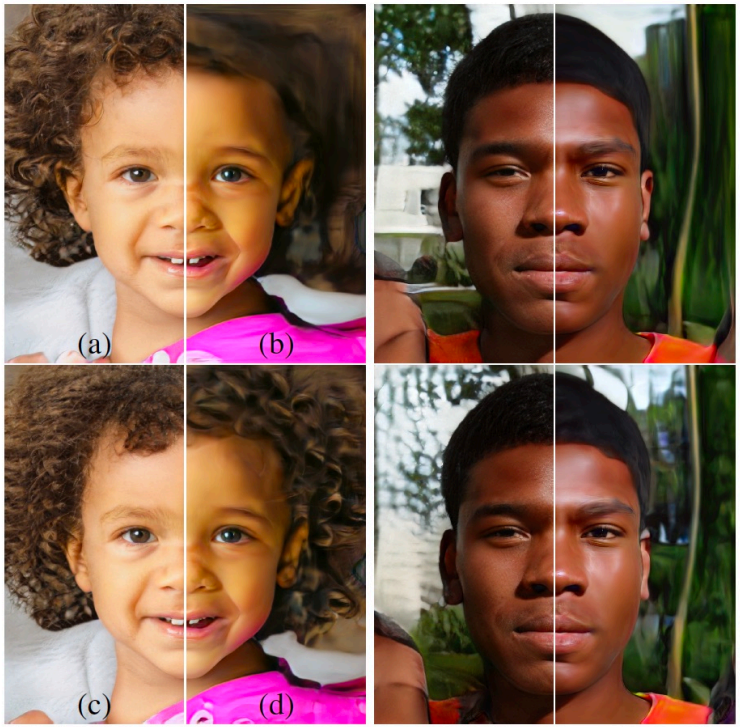
- For coarse resolution (4^2 – 8^2) → high-level aspects such as pose, hair style, face shape, and eyeglasses from source B
- For middle resolution (16^2 – 32^2) → smaller scale facial features, hair style, eyes open/closed from B, while pose, face shape, and eyeglasses from A
- For high resolution (64^2 – 1024^2) → B brings mainly the color scheme and microstructure



StyleGAN: The Effect of Noise

No noise leads to featureless “painterly” look

- (a) Noise is applied to all layers
- (b) No noise
- (c) Noise in fine layers only (64^2 – 1024^2)
- (d) Noise in coarse layers only (4^2 – 32^2)



StyleGAN: Summary

Style-based generator architecture performs better than PGGAN

But a comparison with BigGAN architecture is missing in the paper

In practice, both architectures work well!

Choose between BigGAN and StyleGAN architectures as you like

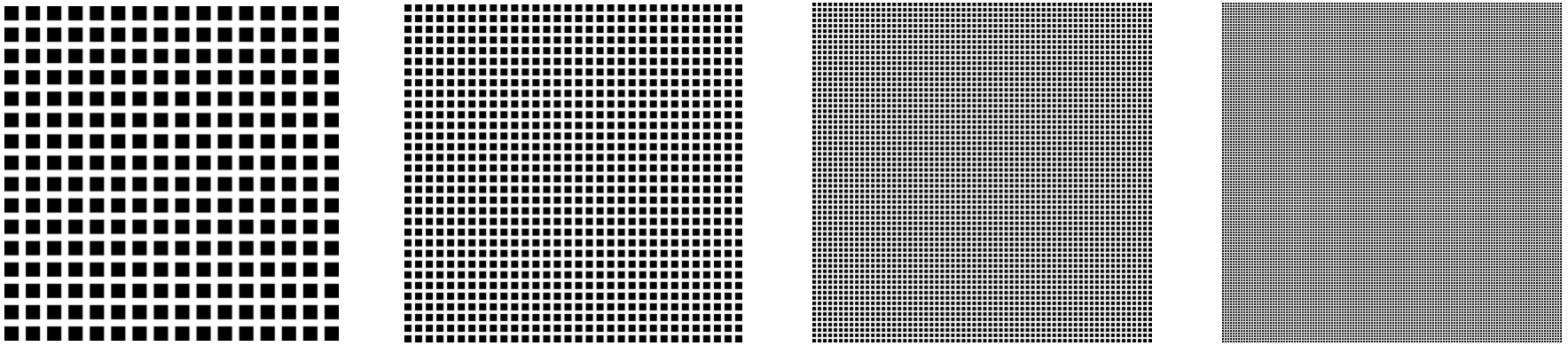
COCO-GAN (COnditional COordinate)

Generator generates images by parts based on the spatial coordinates as the condition

- Discriminator learns to justify realism by global coherence, local appearance and edge-crossing continuity

Achieves SOTA on CelebA, CelebA-HQ (128x128 not 1024x1024 by StyleGAN) and LSUN Bedroom (256x256)

- Novel applications: Extrapolation and Panorama generation



[1] Lin et al. (NTHU), COCO-GAN: Generation by Parts via Conditional Coordinating, ICCV 2019.

COCO-GAN: Approach

Two networks (G, D) and two coordinate systems

- Finer-grained micro for G, coarser-grained macro for D)

Three image sizes

- Full image (real: x, generated: s), macro patches for D (real: x', generated: s') and micro patches for G (real: x'', generated: s'')

G generates micro patches: $s'' = G(z, c'')$

- z: latent code, c'': micro coordinates

D evaluates the reality of macro-patches (assembles of multiple micro-patches)

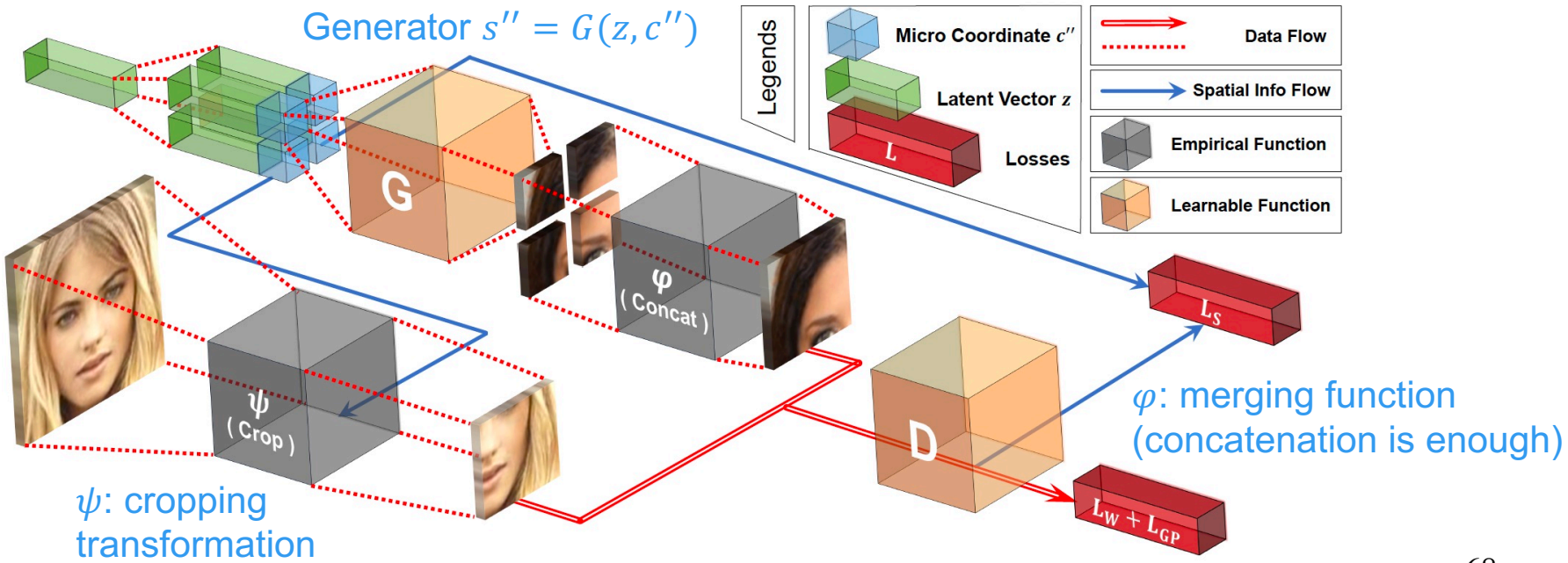
- To fool D, G is forced to consider continuity and coherence of nearby micro-patches

COCO-GAN: Approach

Losses for macro-patch similarity (from Wasserstein GAN)

- L_W: Wasserstein distance loss, L_{GP}: Gradient penalty

Losses for spatial consistency L_s (D predicts the coordinates of macro-patches)



COCO-GAN: Results

The full image is neither shown nor generated during training!

Visually smooth and globally coherent full images without any post-processing

- 1st row: generated full images, 2nd: macro-patches, 3rd: micro-patches

(a) CelebA (N2,M2,S32) (full image: 128×128).

(b) LSUN bedroom (N2,M2,S64) (full image: 256×256).

COCO-GAN: Results

Comparison of FID scores

- The backbone: projection discriminator

Dataset	CelebA 64×64	CelebA 128×128	LSUN Bedroom 64×64	LSUN Bedroom 256×256	CelebA-HQ 1024×1024
DCGAN [26] + TTUR [13]	12.5	-	57.5	-	
WGAN-GP [12] + TTUR [13]	-	-	9.5	-	-
IntroVAE [14]	-	-	-	8.84	-
PGGAN [15]	-	7.30	-	8.34	**7.48**
Proj. *D* [22] (our backbone)	-	19.55	-	-	-
Ours (N2,M2,S32)	**4.00**	**5.74**	**5.20**	**5.99***	9.49*

[1] Miyato & Koyama, cGANs with Projection Discriminator. ICLR 2018.

COCO-GAN: Results

Beyond-boundary image generation

- Extrapolating the learned training coordinate manifold

Panorama generation

- Trained with a cylindrical coordinate system

COCO-GAN: Summary

Interesting idea! Generator generates images by parts!

Strong performance on generation of mid-resolution images

Appealing new applications: beyond-boundary image generation and panorama generation

Conclusion

Scale up the model to have enough capacity and right inductive bias

Choose between BigGAN and StyleGAN architectures for high-resolution

Use self-attention layers if your target images are quite structured

Progressive training helps a lot in terms of convergence speed & stability

Use truncation trick to adjust trade-off between diversity and fidelity of samples