

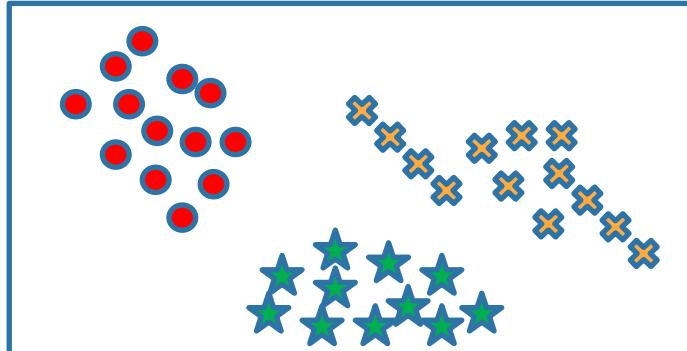
Deep generative methods



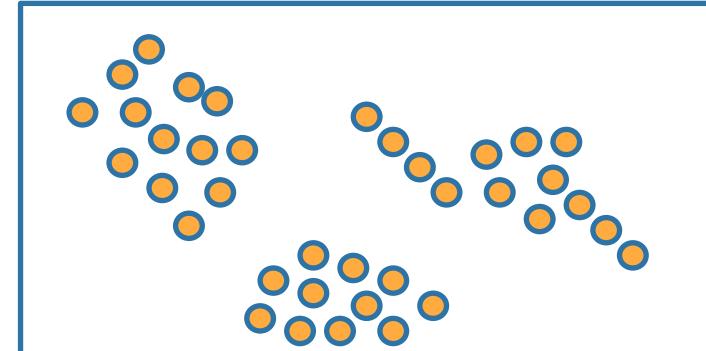
Stéphane Lathuilière
Assistant Professor
Télécom Paris

Types of Learning

- **Supervised (inductive)** learning
 - Given: training data + desired outputs (labels)
- **Unsupervised** learning
 - Given: training data (without desired outputs)



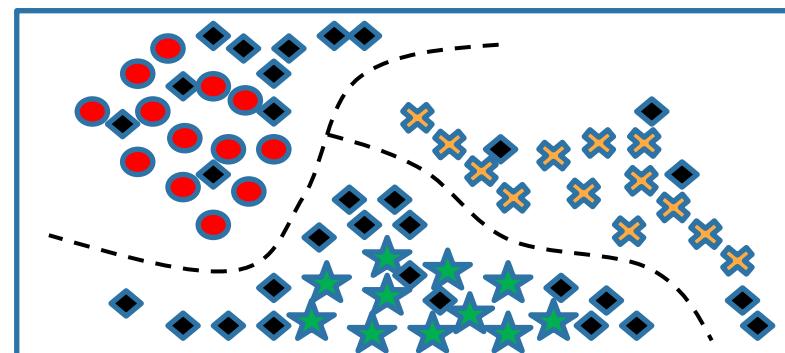
Supervised learning



Unsupervised learning

Types of Learning

- **Supervised (inductive)** learning
 - Given: training data + desired outputs (labels)
- **Unsupervised** learning
 - Given: training data (without desired outputs)
- **Semi-supervised** learning
 - Given: training data + a few desired outputs



Semi-supervised learning

Types of Learning

- **Supervised (inductive)** learning
 - Given: training data + desired outputs (labels)
- **Unsupervised** learning
 - Given: training data (without desired outputs)
- **Semi-supervised** learning
 - Given: training data + a few desired outputs
- **Reinforcement** learning
 - Rewards from sequence of actions

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y) x is data, y is label



→ Cat

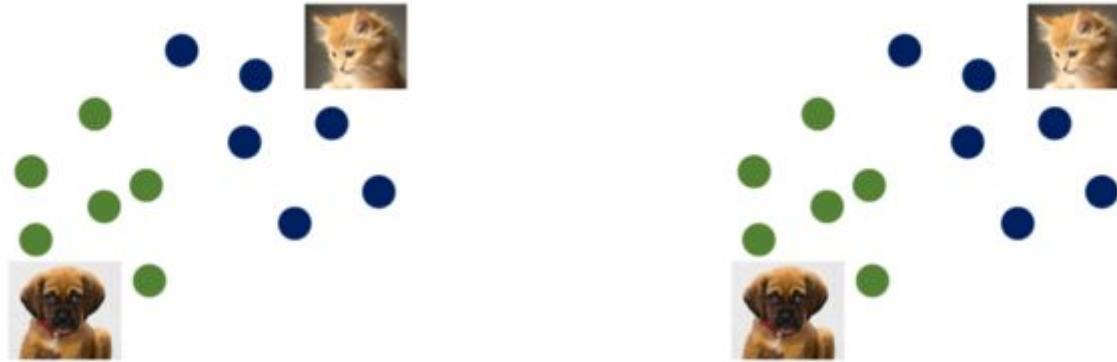
Goal: Learn a *function* to map $x \Rightarrow y$

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Classification

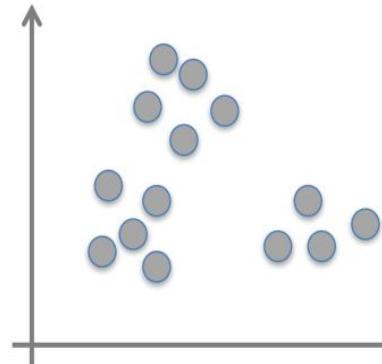
Supervised Learning: Classification

Discriminative vs Generative Algorithms



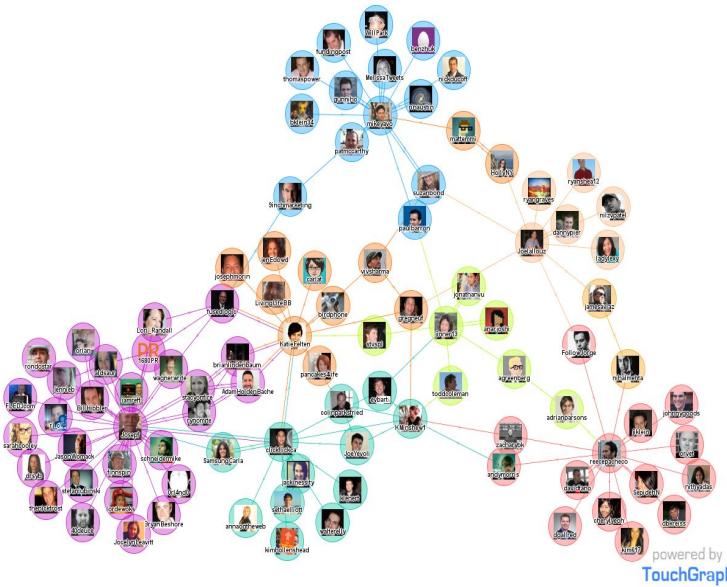
Unsupervised Learning: Clustering

- Given $\mathcal{T} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ (without labels)
output hidden structure behind the \mathbf{x} 's – e.g., clustering



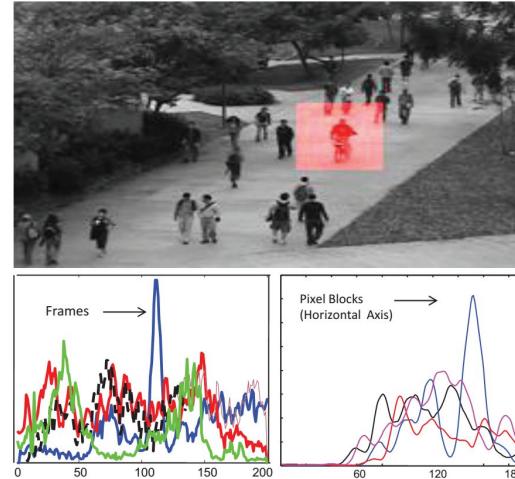
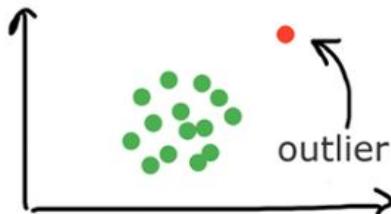
Unsupervised Learning

- Social network analysis



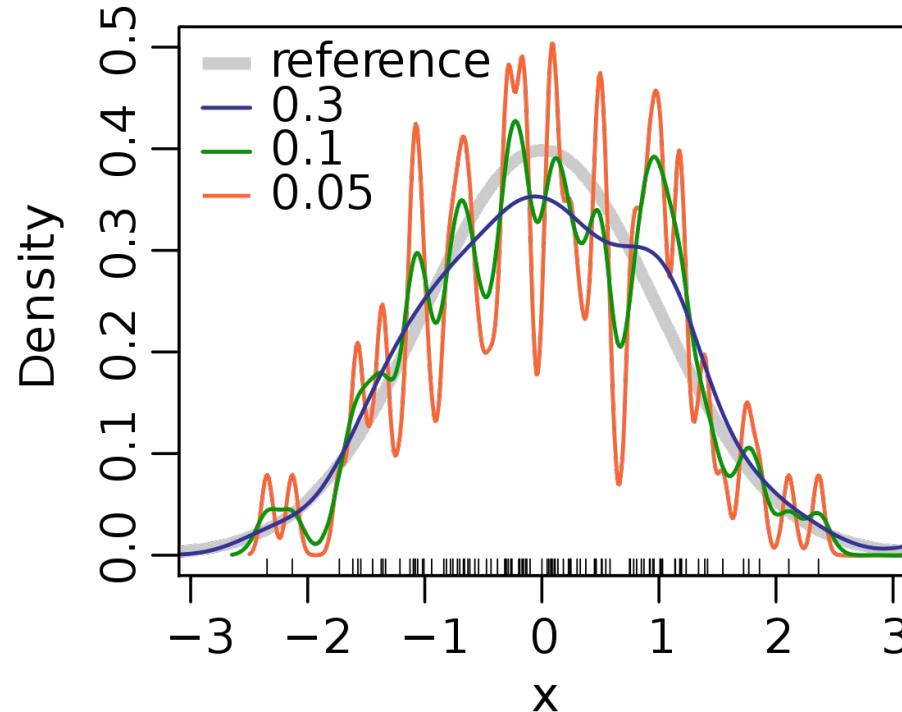
Unsupervised Learning

- **Anomaly detection:** the computer program sifts through a set of events or objects and flags some of them as being unusual or atypical.
- Example: credit card fraud detection, video surveillance.



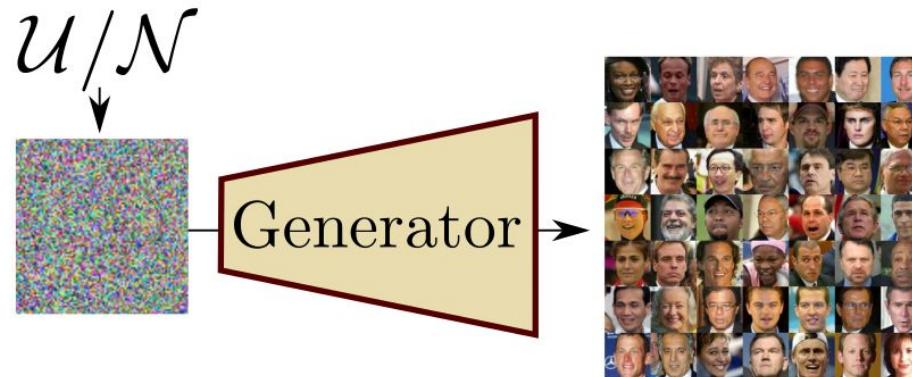
Unsupervised Learning

- Density Estimation: learn a function $p(x): \mathbb{R}^d \rightarrow \mathbb{R}$



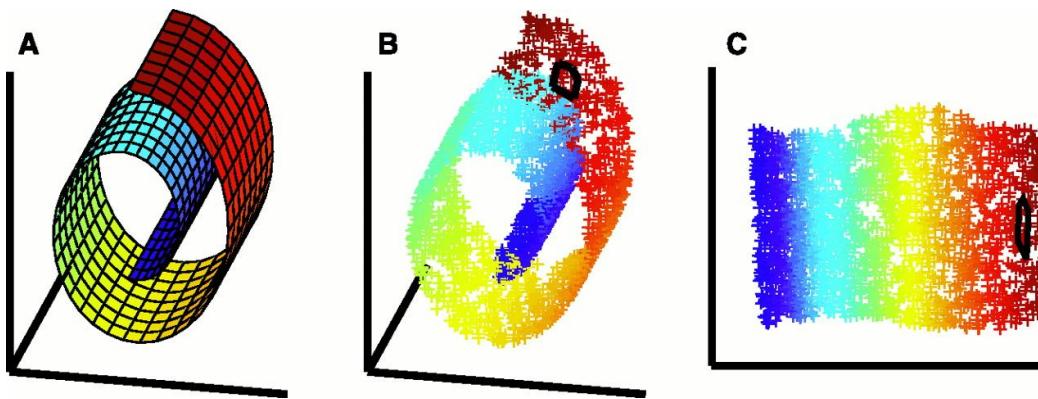
Unsupervised Learning

- Sampling: learn to sample according to $p(x): \mathbb{R}^d \rightarrow \mathbb{R}$



Unsupervised Learning

- Dimensionality Reduction



Supervised vs Unsupervised Learning

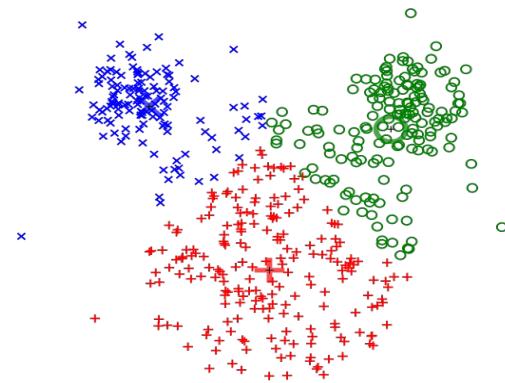
Unsupervised Learning

Data: x

Just data, no labels!

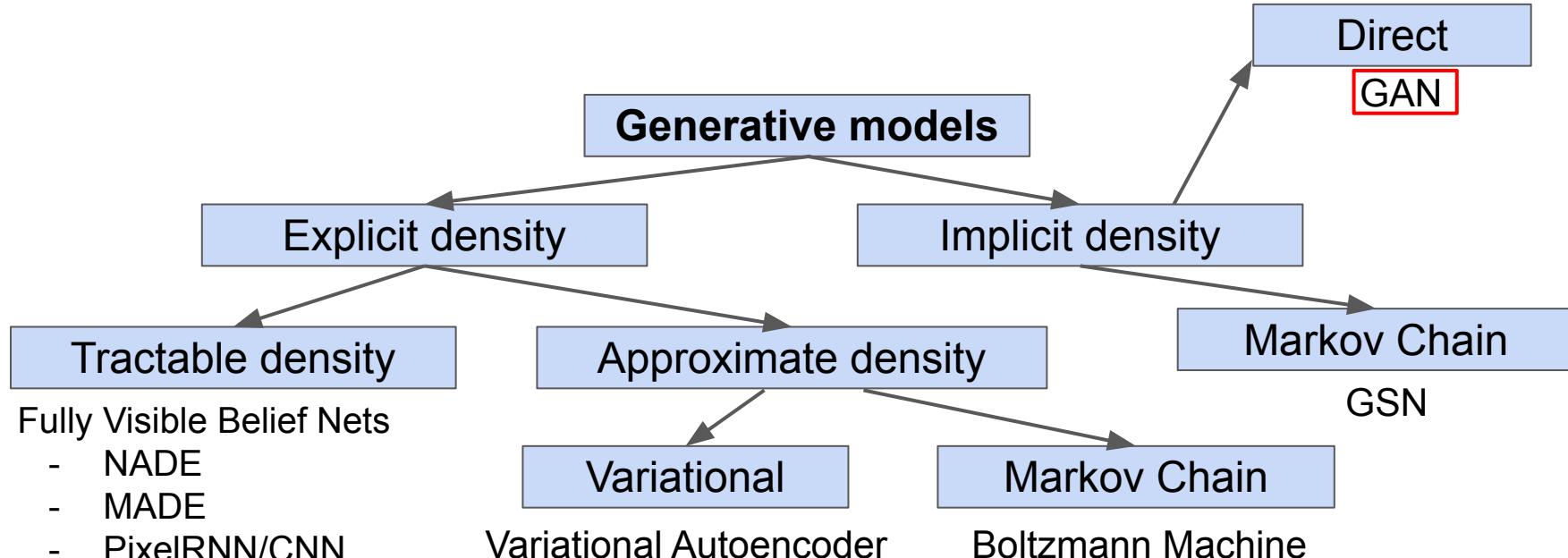
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, **learn a data sampler**, etc.

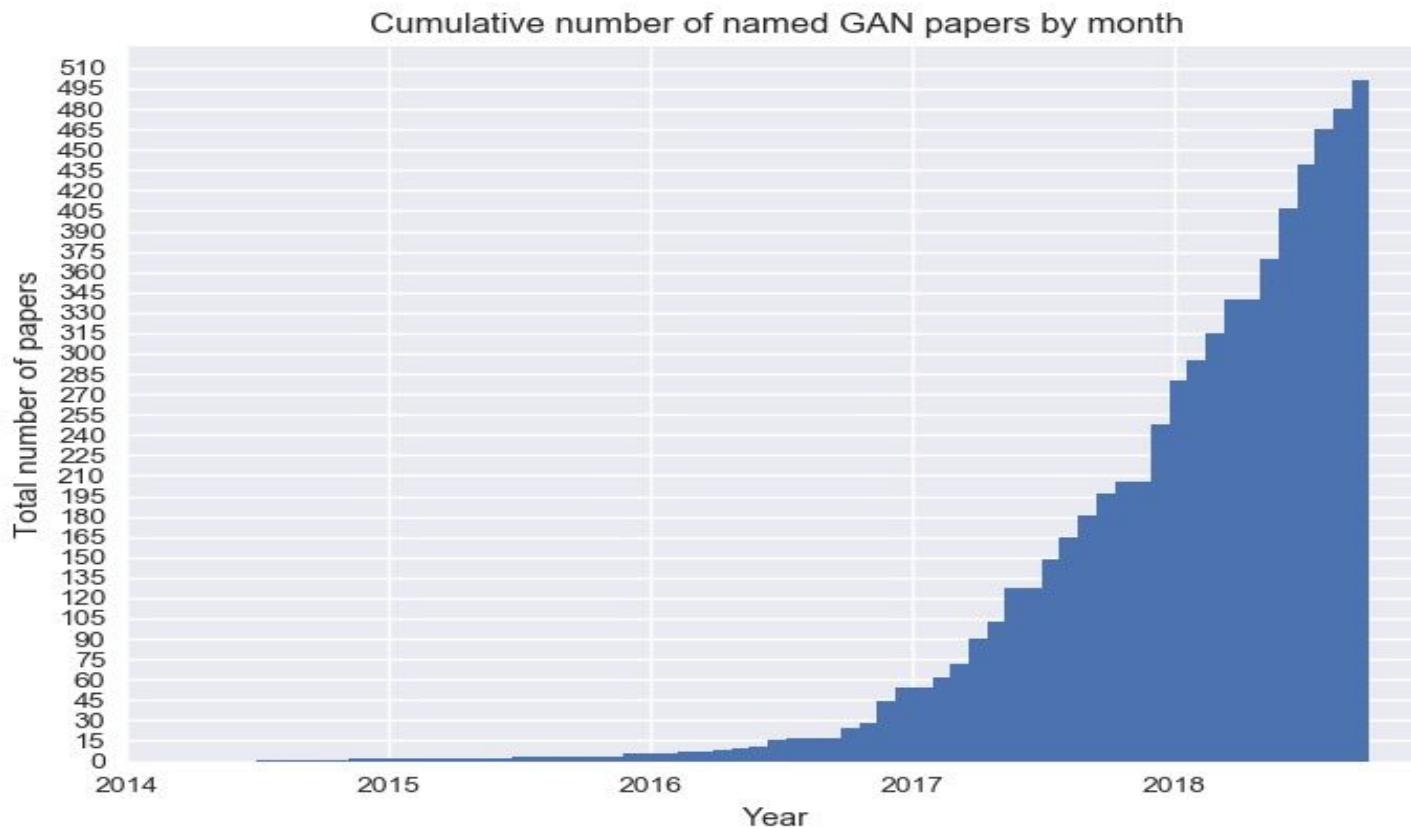


One of the solution for sampling in GAN.

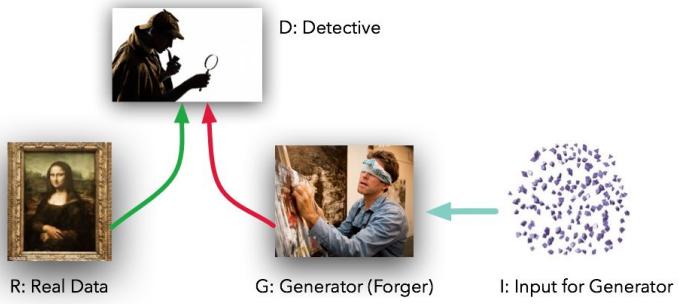
Taxonomy of Generative Models



GAN zoo



<https://github.com/hindupuravinash/the-gan-zoo>



Generative Adversarial Networks (GAN)

Generative Adversarial Networks

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

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

Training GANs: Two-player game

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images

Training GANs: Two-player game

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

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

Training GANs: Two-player game

Minimax objective function:

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

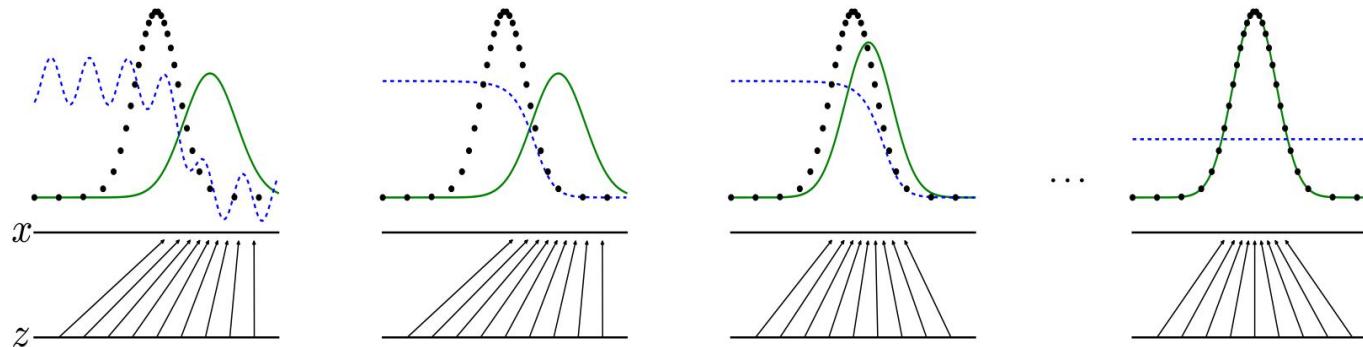
Alternate between:

1. Gradient ascent on discriminator

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

2. Gradient descent on generator

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



GAN Training Algorithm

- Update discriminator
 - Repeat for k steps:
 - Sample mini-batch of noise samples z_1, \dots, z_m and mini-batch of real samples x_1, \dots, x_m
 - Update parameters of D by stochastic gradient ascent on
$$\frac{1}{m} \sum_m [\log D(x_m) + \log(1 - D(G(z_m)))]$$
- Update generator
 - Sample mini-batch of noise samples z_1, \dots, z_m
 - Update parameters of G by stochastic gradient ascent on
$$\frac{1}{m} \sum_m \log D(G(z_m))$$

GAN Results

MNIST digits



Toronto Face Dataset



Nearest real image for
sample to the left

GAN Results

CIFAR-10 (FC networks)



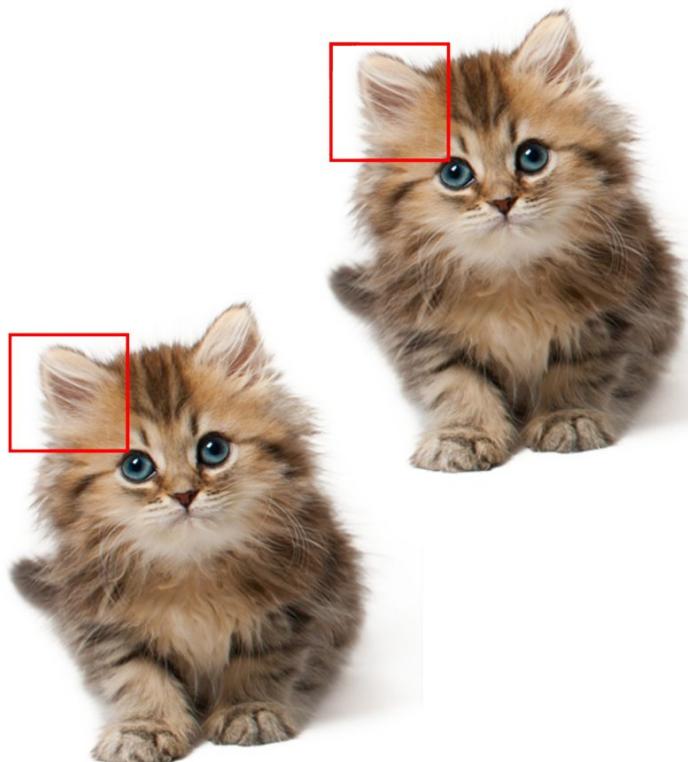
CIFAR-10 (conv networks)



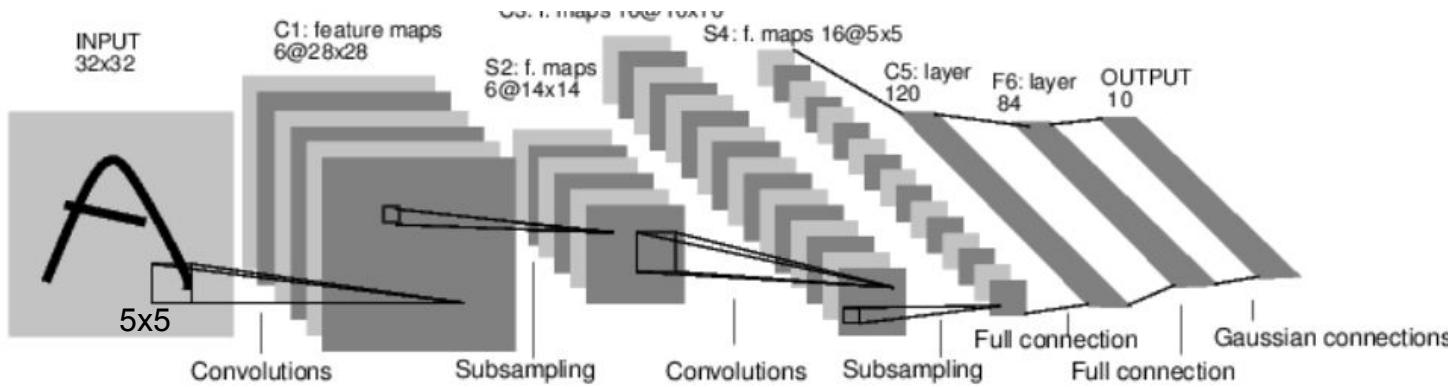
How to improve GAN?

Let's go to basics!

Layers: convolution



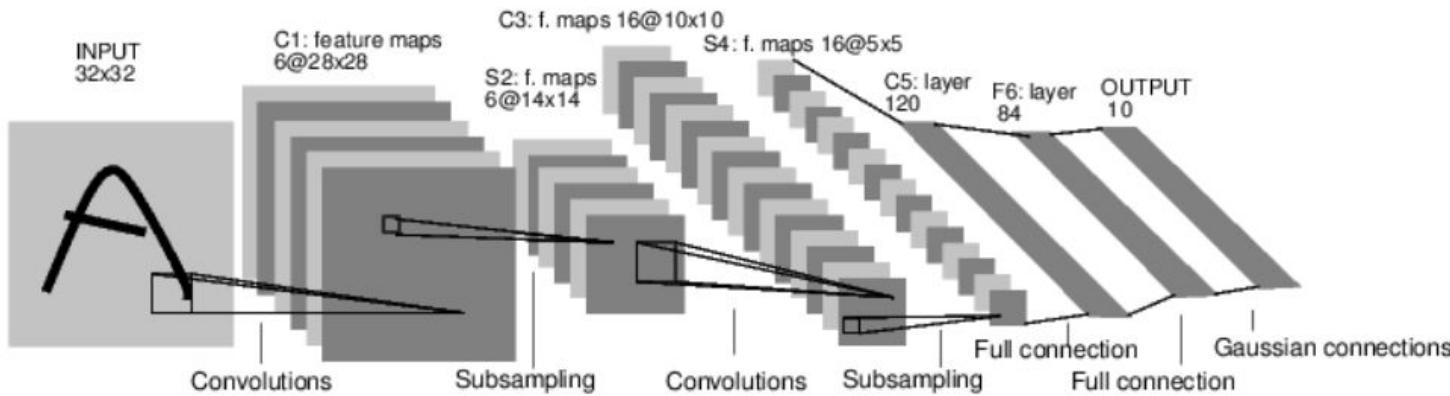
Layers: convolution



First layer:

- Without weight sharing:
- with weight sharing:

Layers: convolution



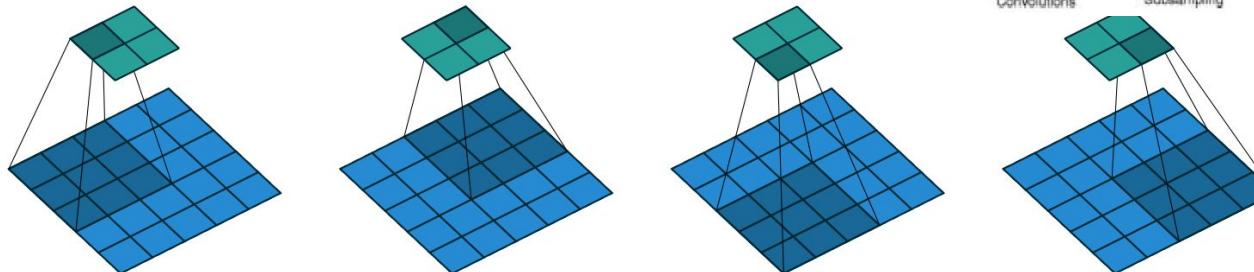
First layer:

- Without weight sharing: $5 \times 5 \times 6 \times 28 \times 28 = 117600$
- with weight sharing: $5 \times 5 \times 6 = 150$

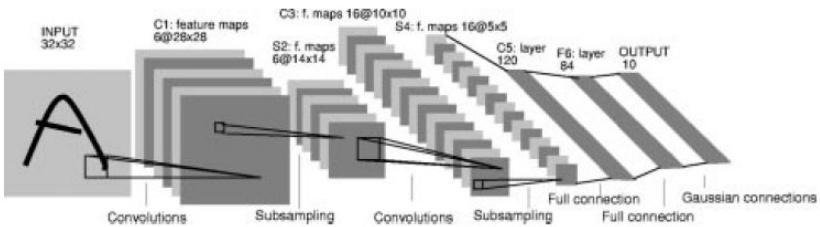
“De-convolution”

Architecture:

- Encoder: ConvNet architecture

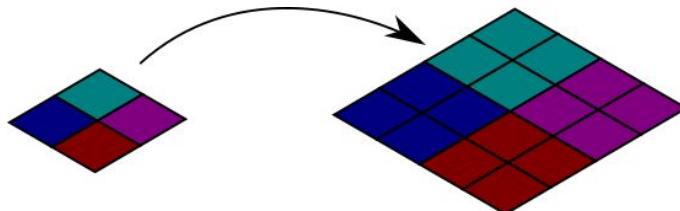


Convolving a 3×3 kernel over a 5×5 input using 2×2 strides



- Decoder: How to go from a vector to an image? How to “deconvolve”?

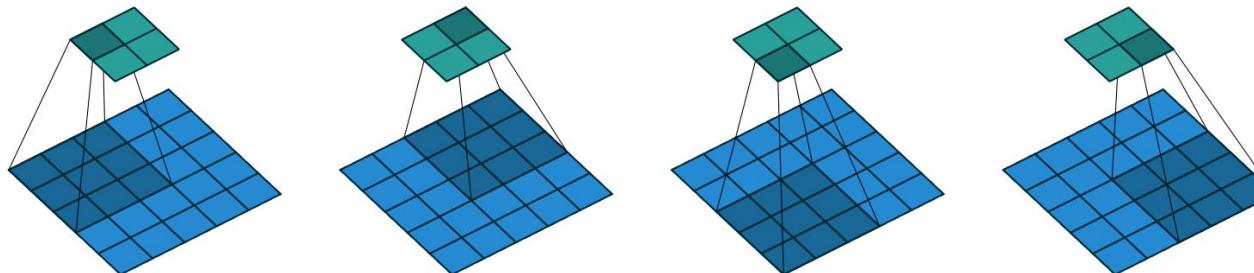
Up-sampling



“De-convolution”

Architecture:

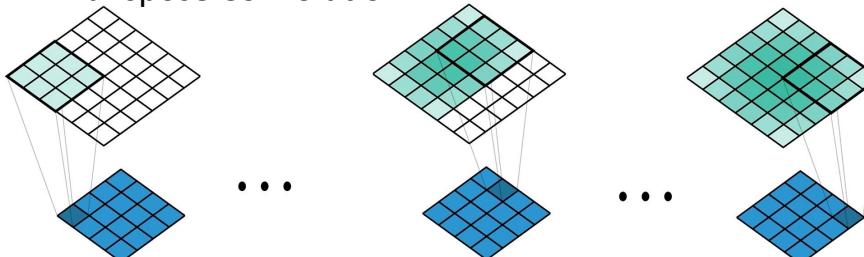
Encoder: ConvNet architecture



Convolving a 3×3 kernel over a 5×5 input using 2×2 strides

Decoder: How to go from a vector to an image? How to “deconvolve”?

Transpose Convolution:



Gradient descent

- Optimization problem: $\min_{\theta} = L_{tot}(\theta)$

- Gradient Descent:

Until convergence: $\theta_{n+1} = \theta_n - \gamma \nabla L(\theta_n)$

- Non convex problem -> Gradient descent does not work!

Gradient descent

- Optimization problem: $\min_{\theta} = L_{tot}(\theta)$
- Gradient Descent:

Until convergence: $\theta_{n+1} = \theta_n - \gamma \nabla L(\theta_n)$

- Non convex problem -> Gradient descent does not work!
- Stochastic Gradient Descent:

○ We split the training in mini-batch: $L_{tot}(\theta) = \sum_b L_b(\theta)$

Until convergence:

For each batch b: $\theta_{n+1} = \theta_n - \gamma \nabla L_b(\theta_n)$

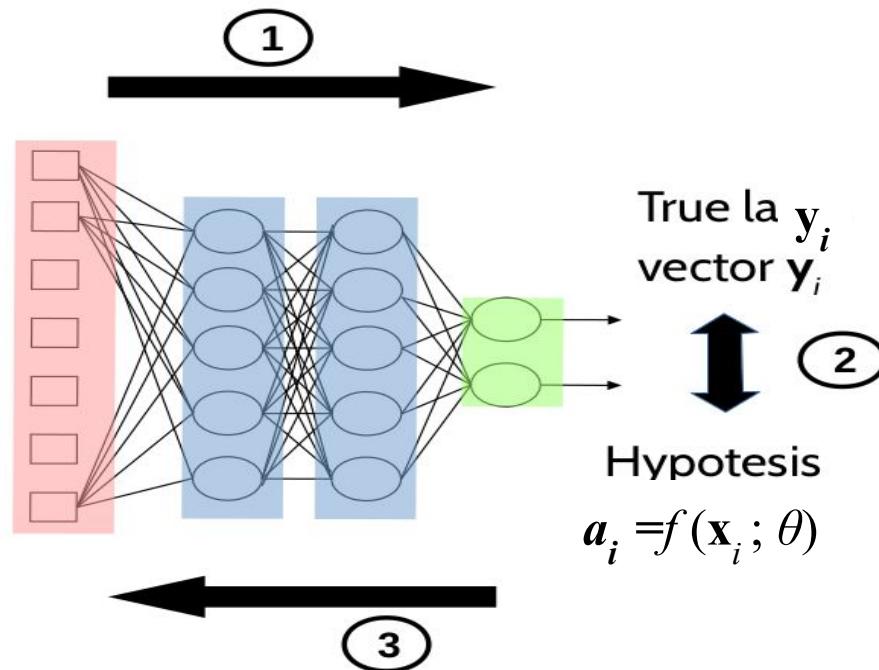
- How to compute the gradient efficiently?

1986 – Backpropagation

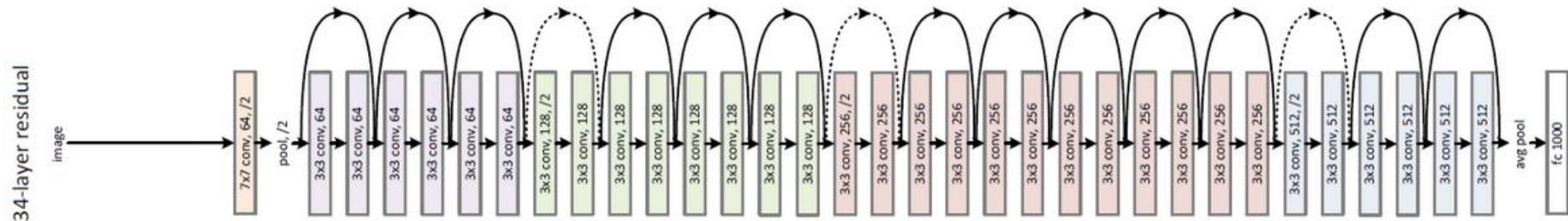
- Backpropagation revitalized the field
- Learning MLP for complicated functions can be solved
- Efficient algorithm which processes “large” training sets
- Allowed for complicated neural network architectures
- Today backpropagation is still at the core of neural network training

Backpropagation in a Nutshell

1. **Forward propagation:** sum inputs, produce activations, feed-forward
2. **Error estimation.**
3. **Back propagate** the error signal and used it to update weights

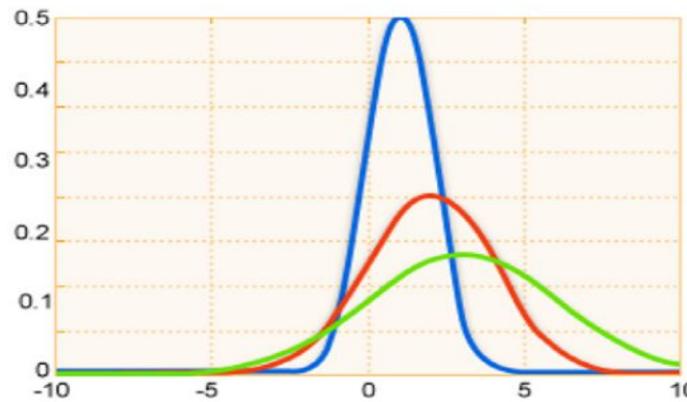


Layers: Batch-normalization

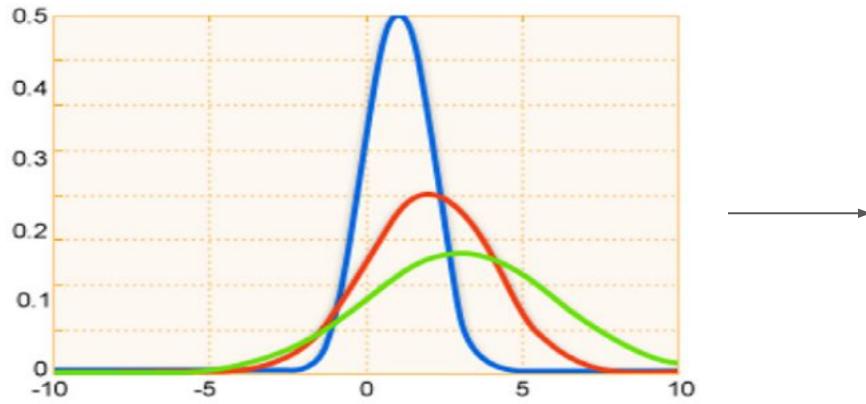


Parameter update:

$$\theta_{n+1} = \theta_n - \gamma \nabla L_b(\theta_n)$$



Layers: Batch-normalization



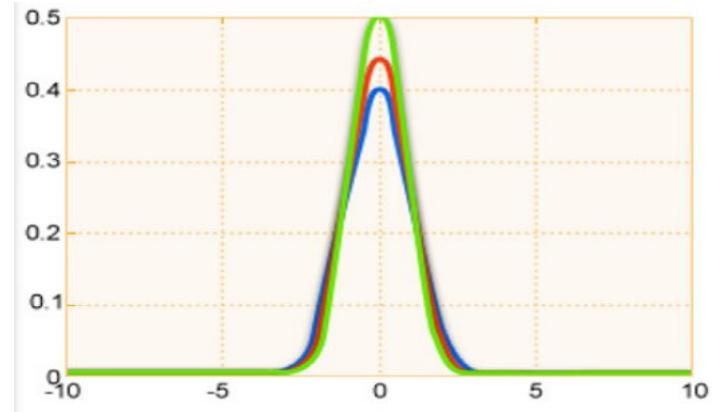
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
Sergey Ioffe, Christian Szegedy

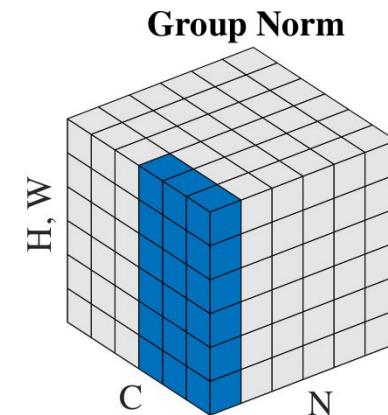
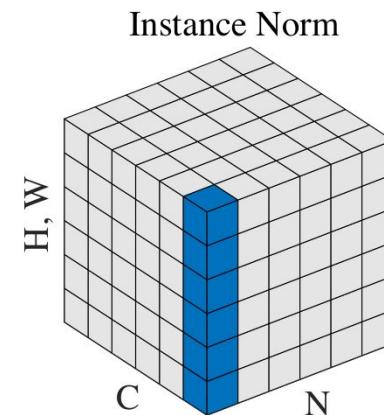
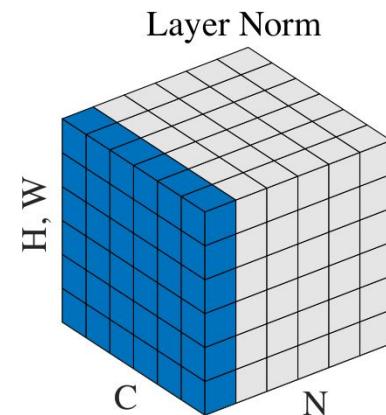
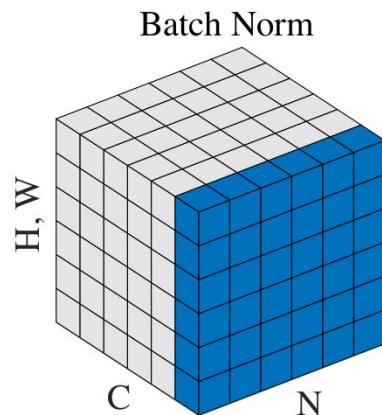
Layers: Batch-normalization

Advantages:

- Faster training
- Better optimization (lower final loss)
- Better generalization
- Specific usages in:
 - few-shot learning
 - domain adaptation
 - generation

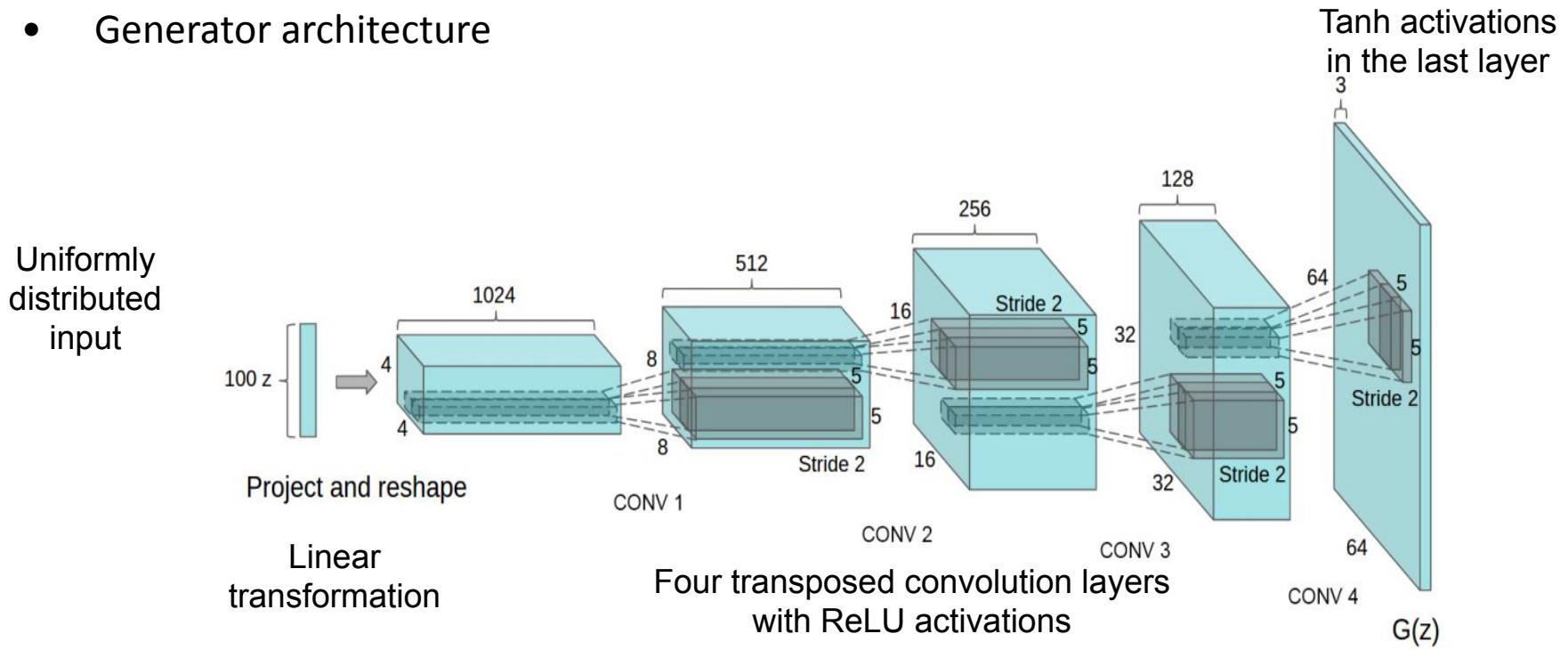
Layers: Other normalizations

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$



DCGAN

- Propose principles for designing convolutional architectures for GANs, first model for large resolution image generation
- Generator architecture



DCGAN

- Discriminator architecture
 - No pooling, only strided convolutions
 - Use Leaky ReLU activations (sparse gradients cause problems for training)
 - Use only one FC layer before the softmax output
 - Use batch normalization after most layers
(in the generator also)

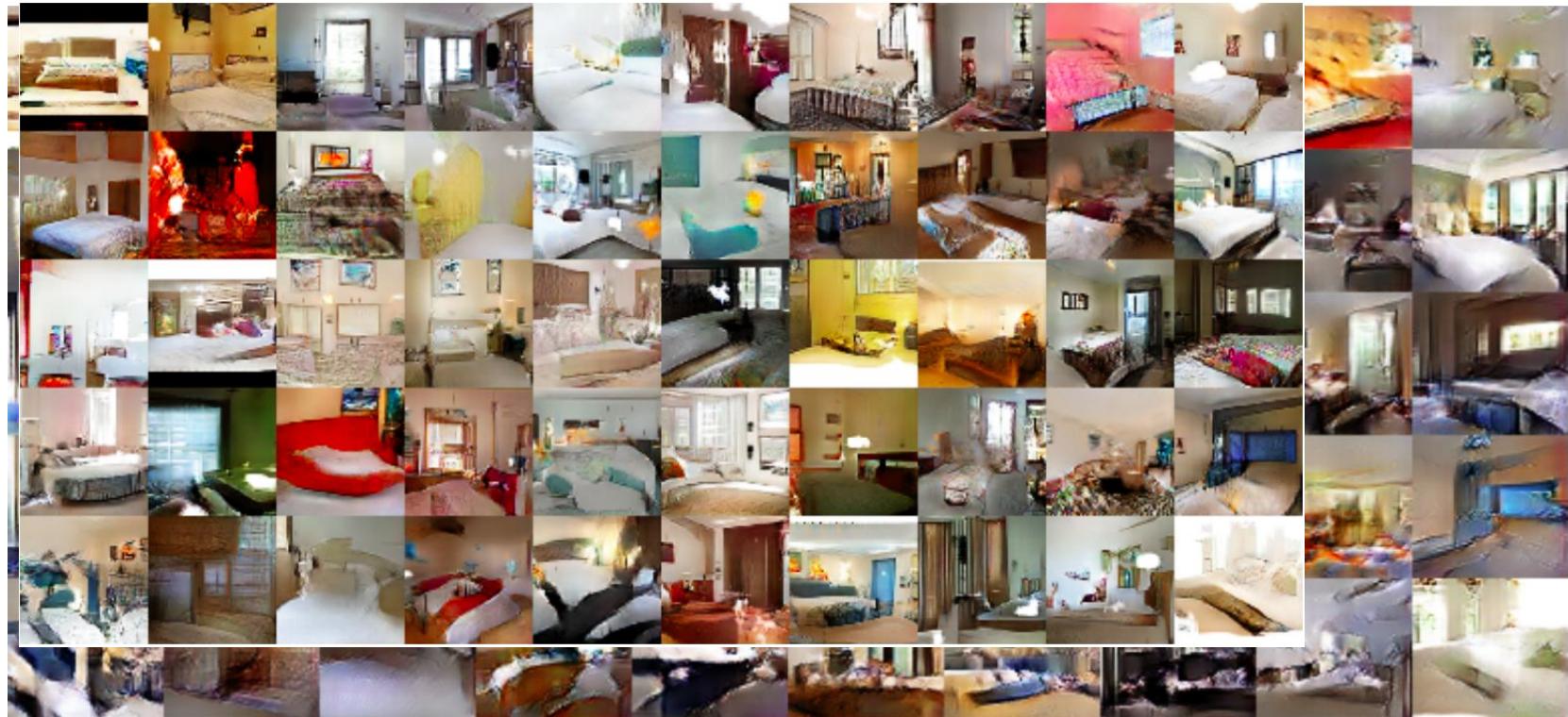
DCGAN results

Generated bedrooms after one epoch



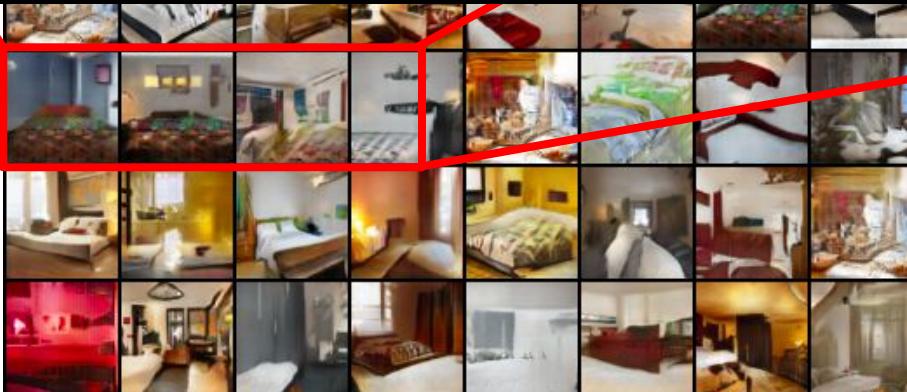
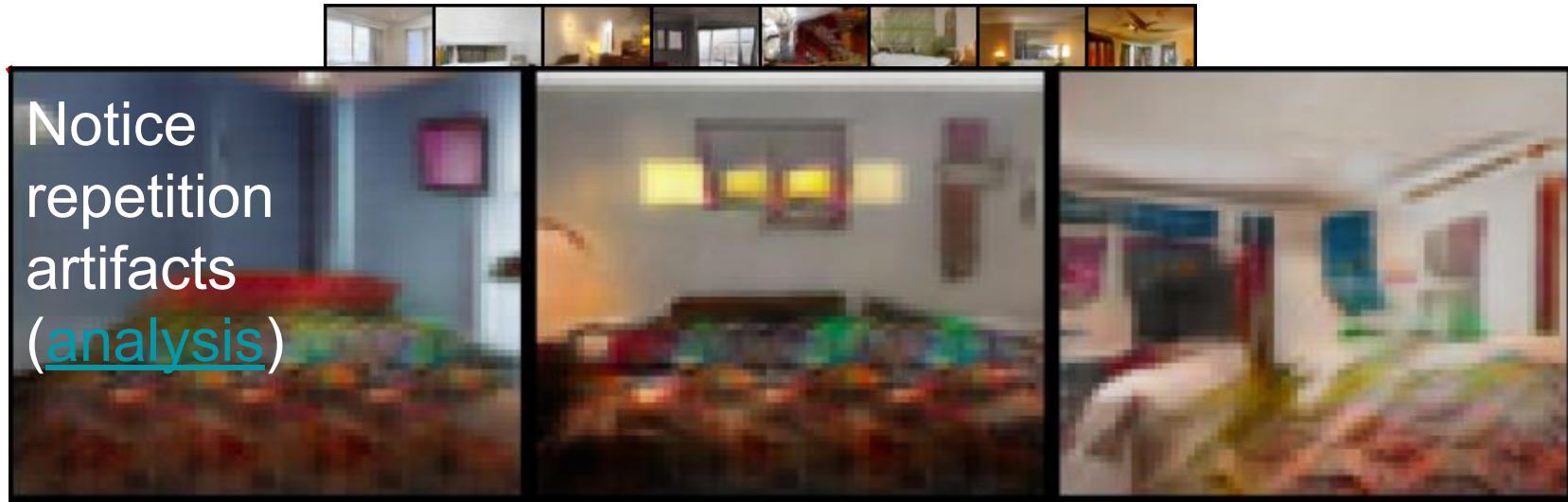
DCGAN results

Generated bedrooms after five epochs

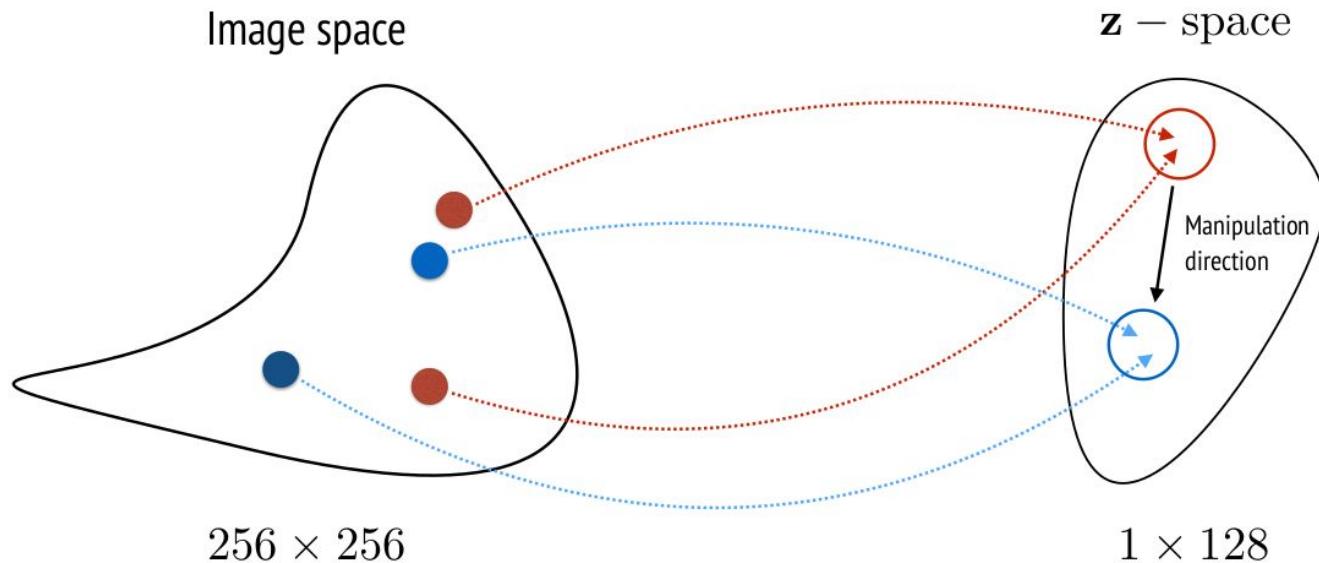


DCGAN results

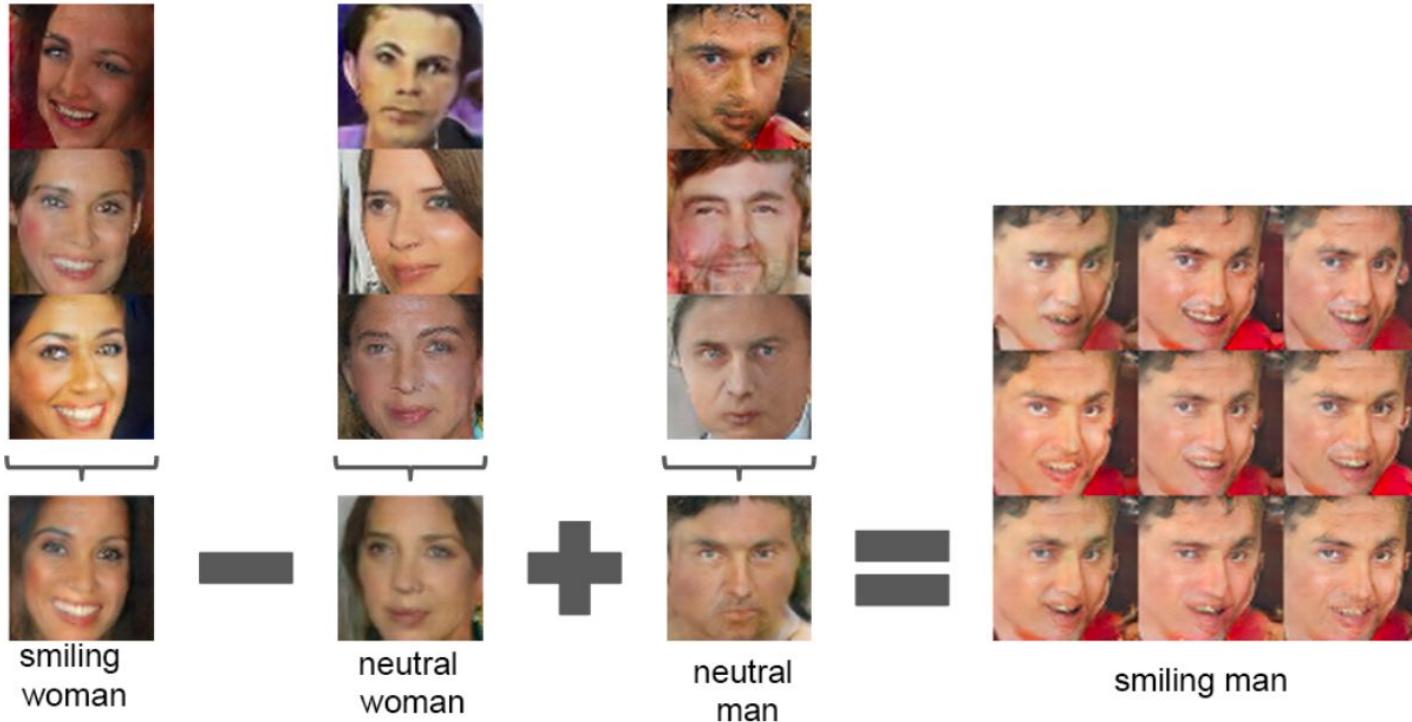
Generated bedrooms from reference implementation



DCGAN results

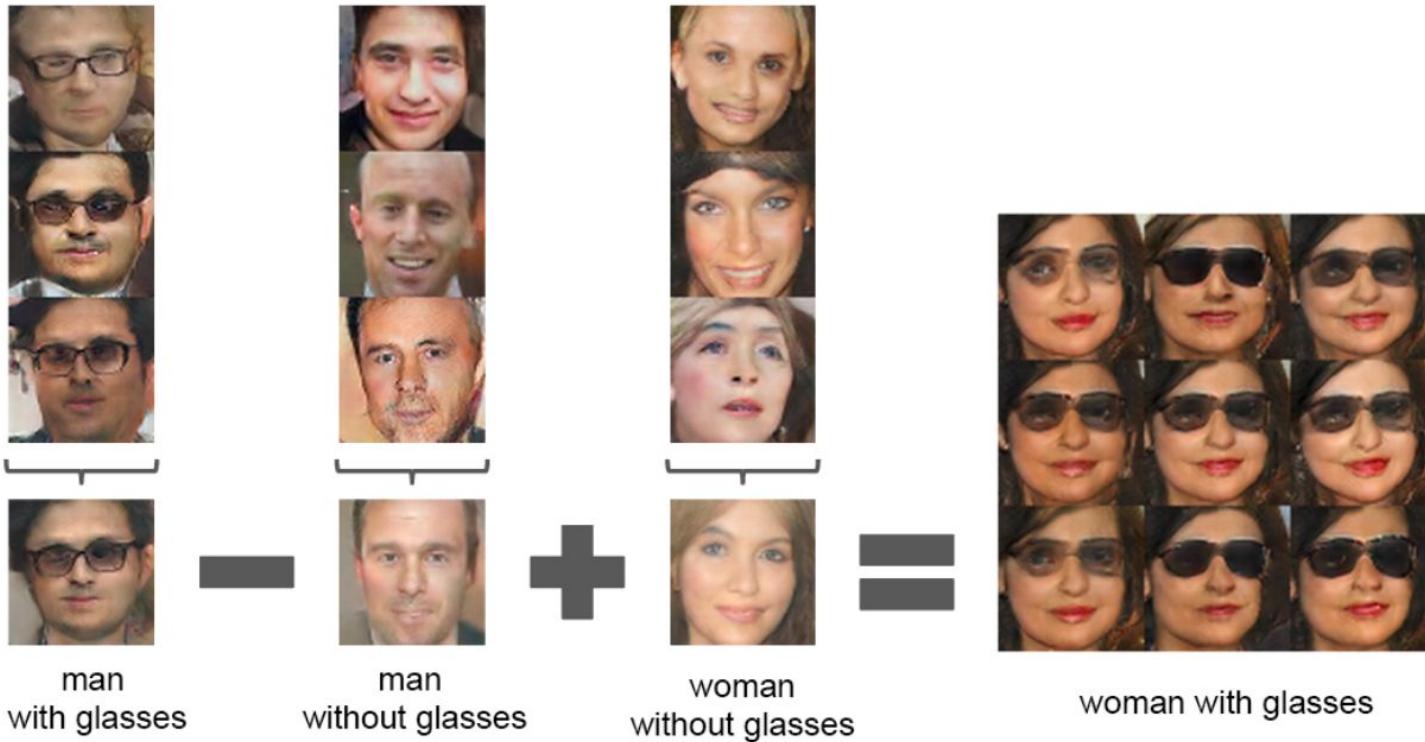


DCGAN results



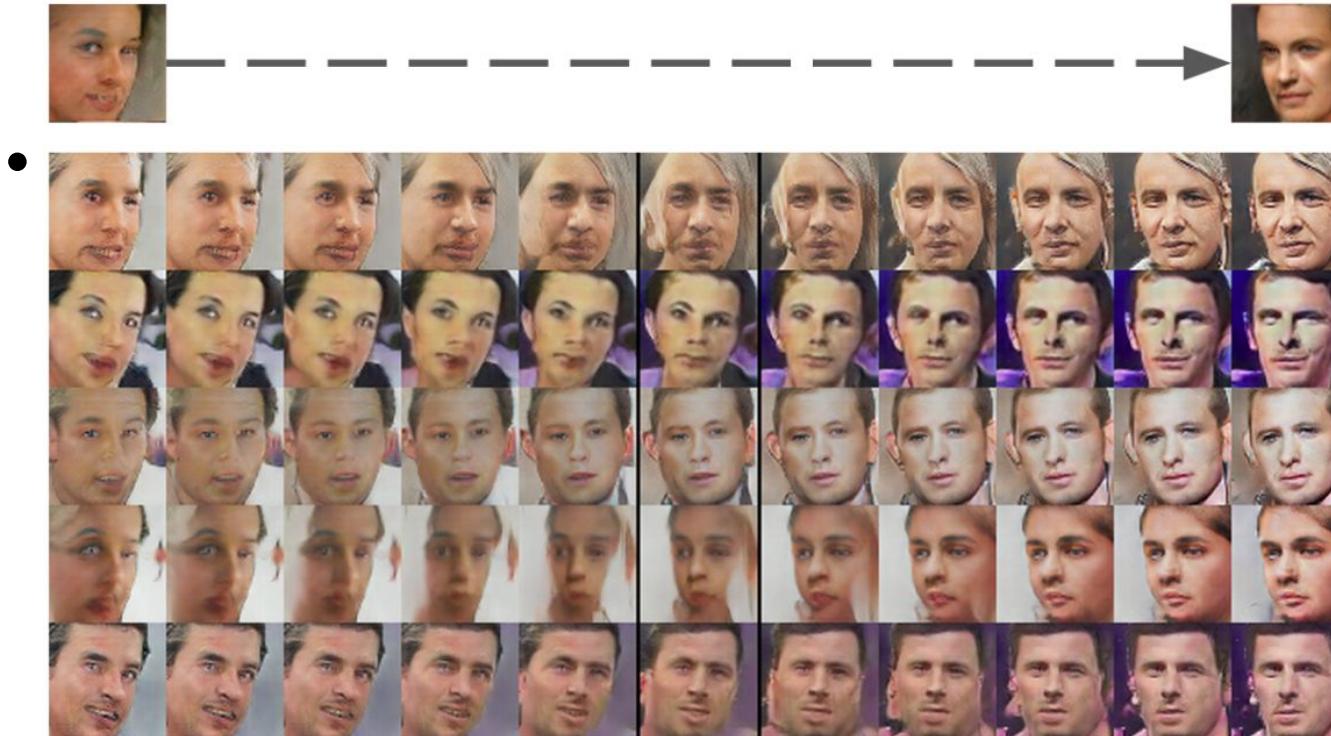
For each column, the Z vectors of samples are averaged. Arithmetic was performed on the mean vectors creating a new vector Y . The center sample on the right hand side is produced by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale ± 0.25 was added to Y to produce the 8 other samples.

DCGAN results



First model to bring the concept of **disentanglement**.

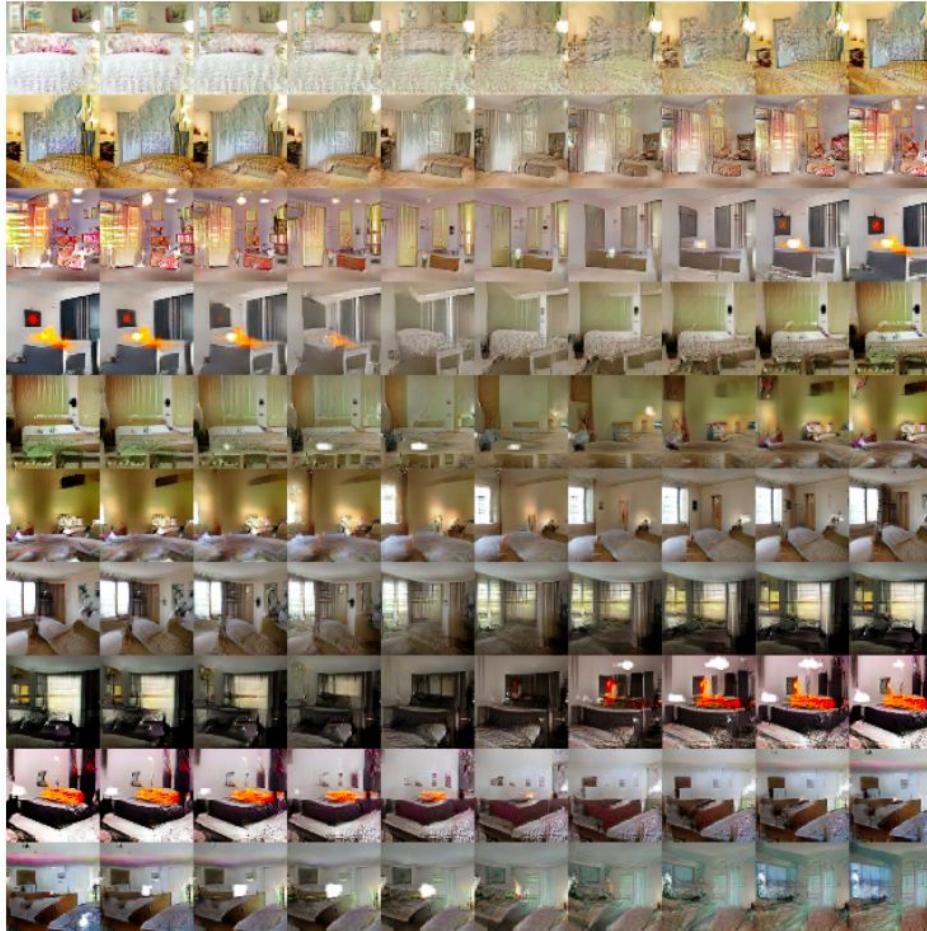
DCGAN results



A "turn" vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

DCGAN results

Interpolation between different points in the z space



Evaluate GAN performance

- Showing pictures of samples is not enough, especially for simpler datasets like MNIST, CIFAR, faces, bedrooms, etc.
- We cannot directly compute the likelihoods of high-dimensional samples (real or generated) or compare their distributions
- Many GAN approaches claim mainly to improve stability, which is hard to evaluate
- For discussion, see [Ian Goodfellow Twitter thread](#)

Evaluate GAN performance

- Human performance

Instructions



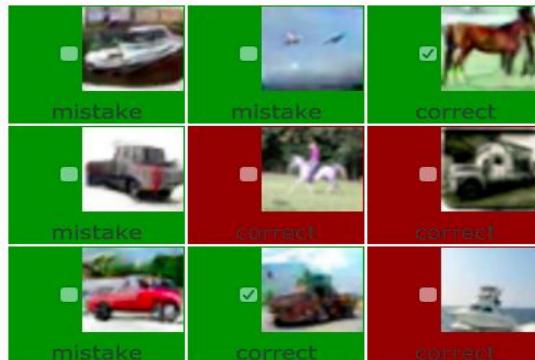
Examples of real images



Examples of images generated by a computer

We marked with green the images generate by computer (YOU SHOULD SET CHECKBOX FOR ALL OF THEM)
and with red all coming from scans of digits (DON'T SET CHECKBOX ON ANY OF THEM).

Please press Next.

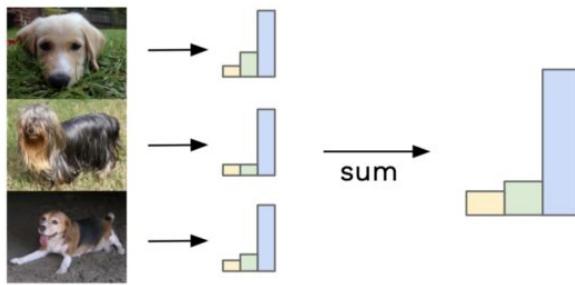


Your score on this question is 5/9

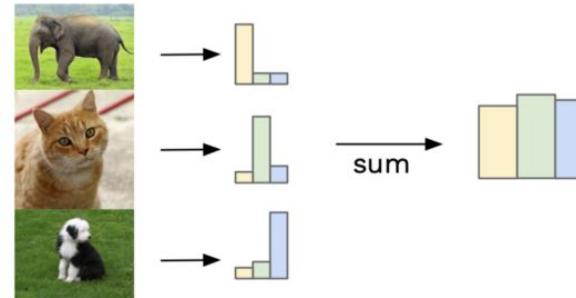
Inception Score

- Key idea: generators should produce images with a **variety of recognizable object classes**
- Two criteria:
 - A human, looking at a separate image, would be able to confidently determine what's in there (**saliency**).
 - A human, looking at a set of various images, would say that the set has lots of different objects (**diversity**).

Similar labels sum to give focussed distribution



Different labels sum to give uniform distribution



Inception Score

- Key idea: generators should produce images with a **variety of recognizable** object classes
- Defined as:

$$\text{IS}(G) = \exp \left(\mathbb{E}_{\mathbf{x} \sim p_a} D_{KL} (p(y|\mathbf{x}) \parallel p(y)) \right),$$

where $p(y|x)$ is the posterior label distribution returned by an image classifier (e.g., InceptionNet) for sample x

- If the image contains a recognizable object, entropy of $p(y|x)$ should be low
- If generator generates images of diverse objects, the marginal distribution $p(y)$ should have high entropy
- **Disadvantage:** a GAN that simply memorizes the training data (overfitting) or outputs a single image per class (mode dropping) could still score well

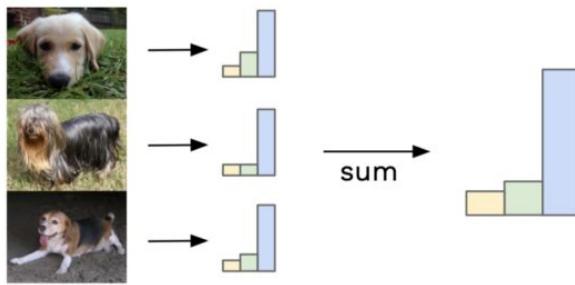
Inception Score

- Key idea: generators should produce images with a **variety of recognizable** object classes
- Defined as:

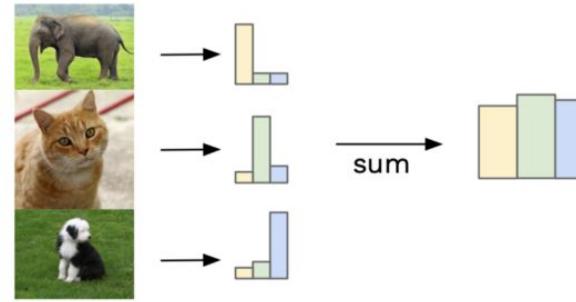
$$\text{IS}(G) = \exp \left(\mathbb{E}_{\mathbf{x} \sim p_a} D_{KL} (p(y|\mathbf{x}) \parallel p(y)) \right),$$

where $p(y|x)$ is the posterior label distribution returned by an image classifier (e.g., InceptionNet) for sample x

Similar labels sum to give focussed distribution



Different labels sum to give uniform distribution



Fréchet Inception Distance

- **Key idea:** fit simple distributions to statistics of feature activations for real and generated data; estimate divergence parametrically
 - Pass generated samples through a network (InceptionNet), compute activations for a chosen layer
 - Estimate multivariate mean and covariance of activations, compute Fréchet distance to those of real data

$$\text{FID}(x, g) = \|\mu_x - \mu_g\|_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{\frac{1}{2}}),$$

- **Advantages:** correlated with visual quality of samples and human judgment
- **Disadvantage:** cannot detect overfitting

Issues with GAN training

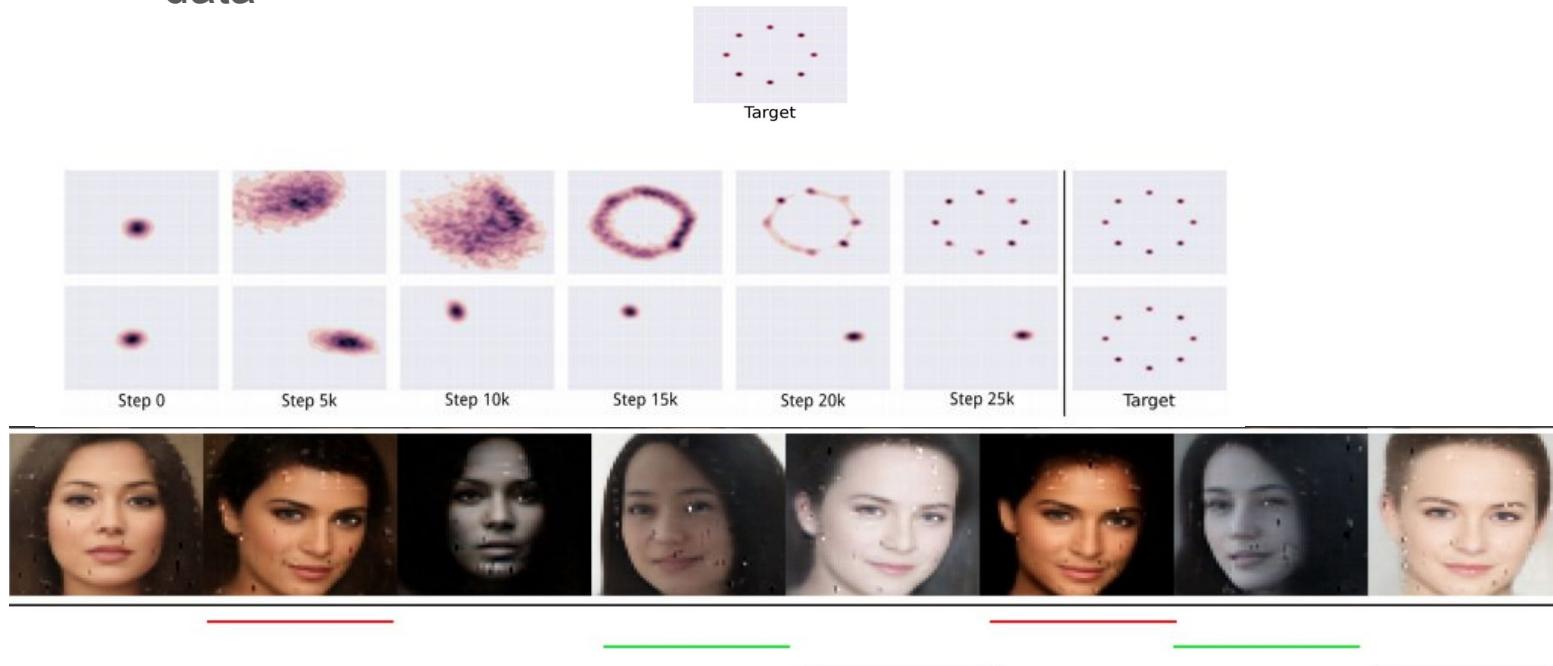
- Stability
 - Parameters can oscillate or diverge, generator loss does not correlate with sample quality
 - Behavior very sensitive to hyper-parameter selection
 - Proposed several improvements, mostly changing the loss function

Tutorial on GAN: summary <https://arxiv.org/pdf/1701.00160.pdf>

https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Issues with GAN training

- Mode collapse
 - Generator ends up modeling only a small subset of the training data



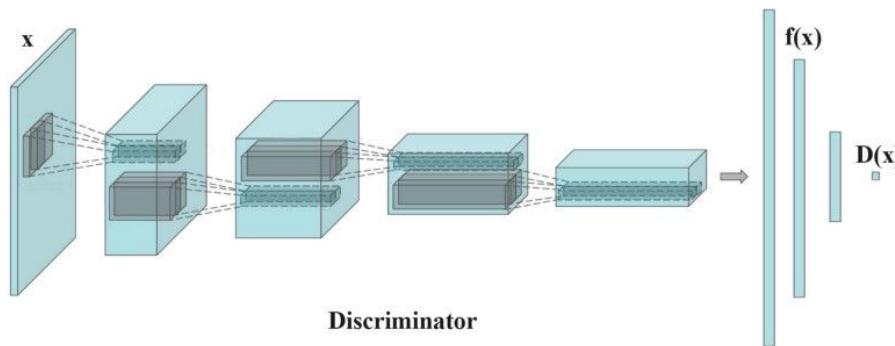
Training Tricks

- Feature matching
- Use virtual batch norm
- Mini-batch discrimination (avoid mode collapsing by looking at more samples)
- Use label informations in the discriminator ($C+1$ categories)
- Label smoothing
- Historical averaging (introducing an additional cost function)

Training Tricks

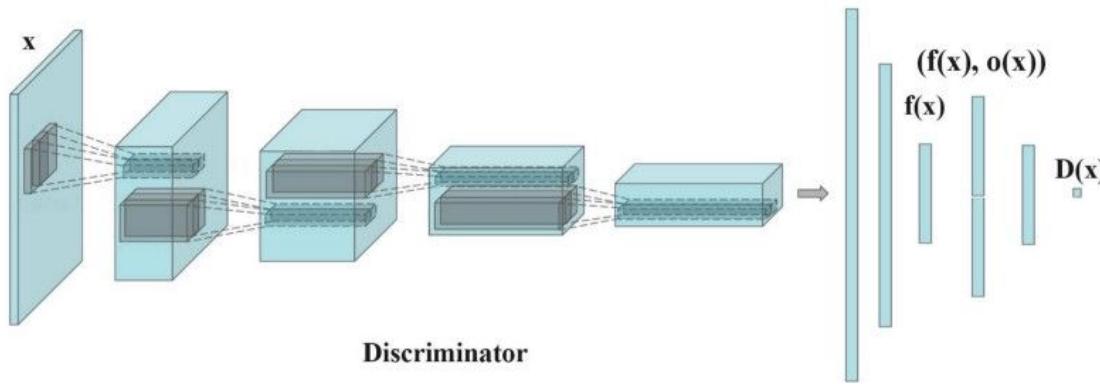
- Feature matching
- Issue:
 - The generator tries to find the best image to fool the discriminator.
 - The “best” image keeps changing when both networks counteract their opponent.
 - The model does not converge and mode collapses.
- Add a loss for feature matching (expands the goal from beating the opponent to matching features in real images)

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



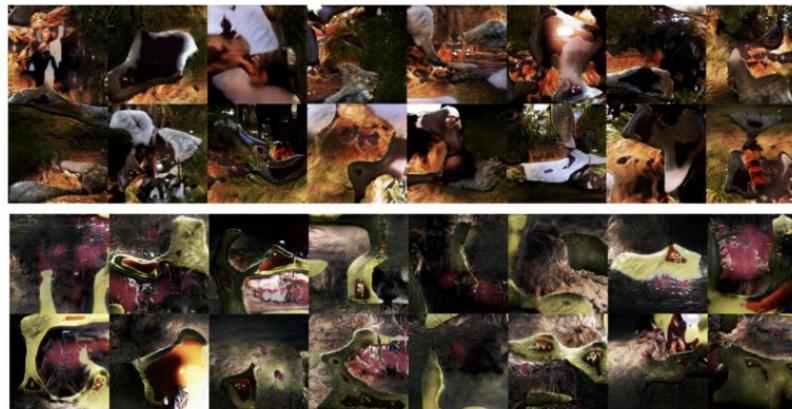
Training Tricks

- Mini-batch discrimination
- Compute the similarity of the image x with images in the same batch. Append the similarity $o(x)$ in one of the dense layers in the discriminator to classify whether this image is real or generated.
- If the mode starts to collapse, the similarity of generated images increases. The discriminator can use $o(x)$ to detect generated images and penalize the generator if mode is collapsing.



Training Tricks

- Virtual batch norm
- Issue: Due to batch norm the generated images are not independent
- Idea: samples a reference batch before the training to compute the normalization parameters (μ and σ) and combine a reference batch with the current batch to compute the normalization parameters.

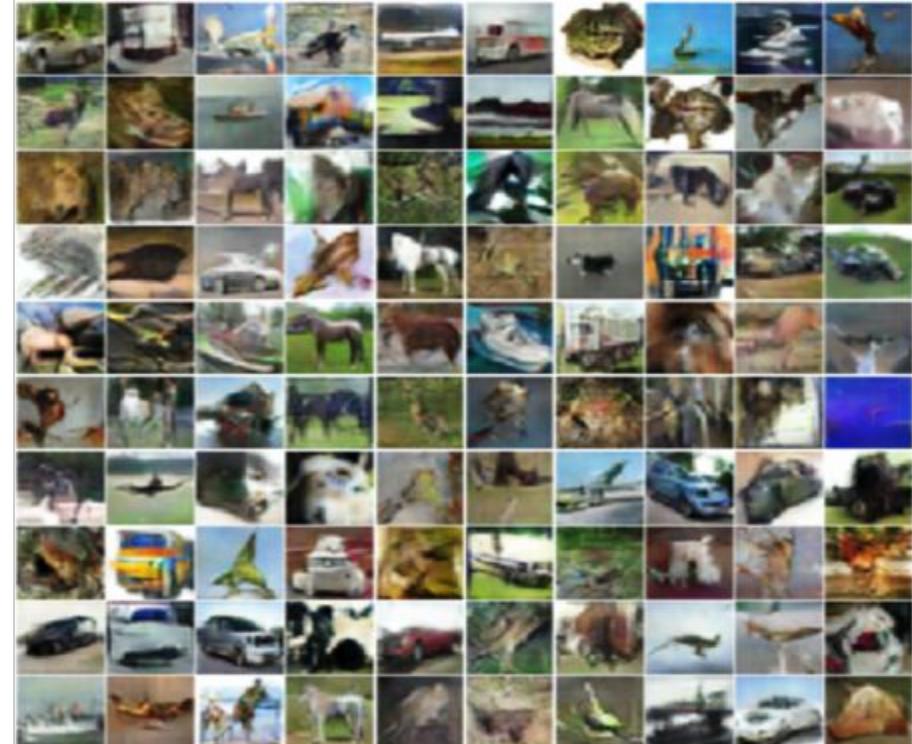
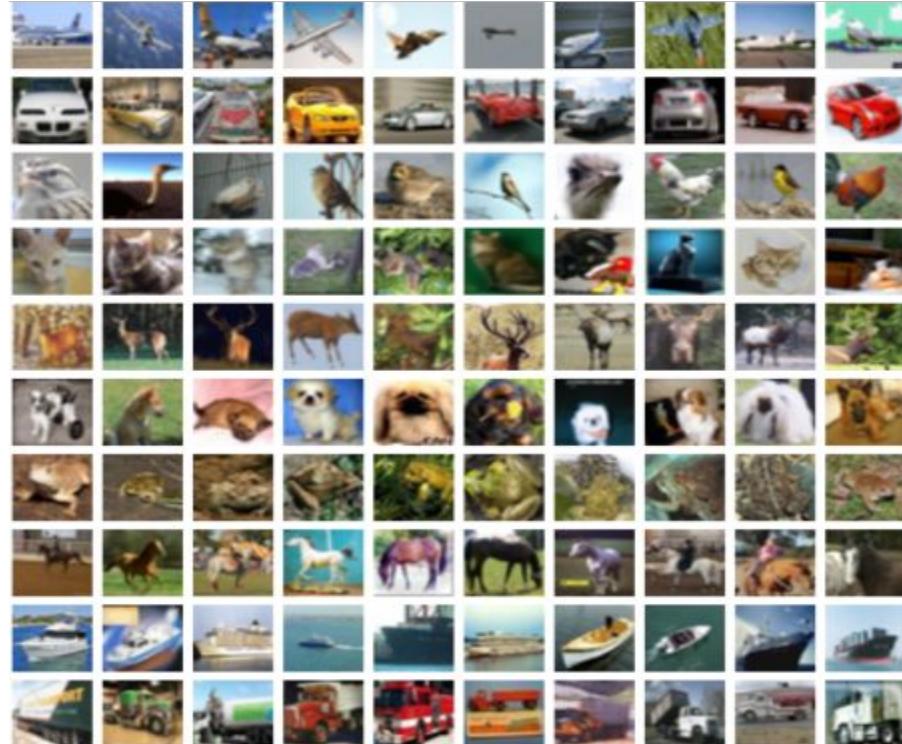


Training Tricks

- Feature matching
- Use virtual batch norm
- Mini-batch discrimination (avoid mode collapsing by looking at more samples)
- Use label informations in the discriminator (C+1 categories)
- Label smoothing
- Historical averaging (introducing an additional cost function)

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

GAN Results



Original CIFAR-10 vs. Generated CIFAR-10 samples

GAN Results

- ImageNet Results (128x128 pixels)

DCGAN

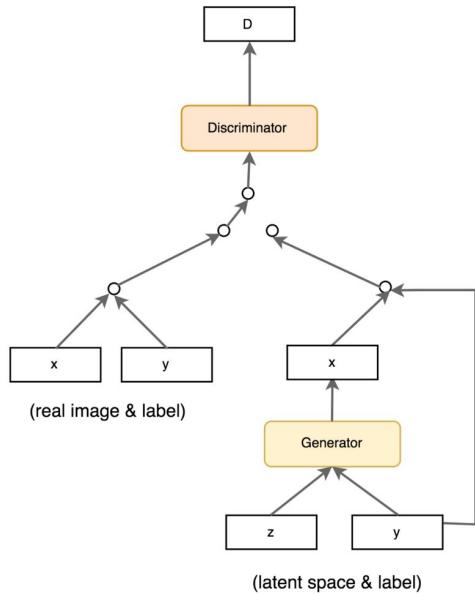


Improved DCGAN



CGAN & InfoGAN

- Conditional GAN (CGAN)
 - Same loss as the original GAN
 - Use label info (1-hot encoding) both in generator and discriminator
 - Label as extension to the latent space



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

CGAN & InfoGAN

- Info GAN (label not given, treated as latent factor):
- Information Maximizing Generative Adversarial Nets

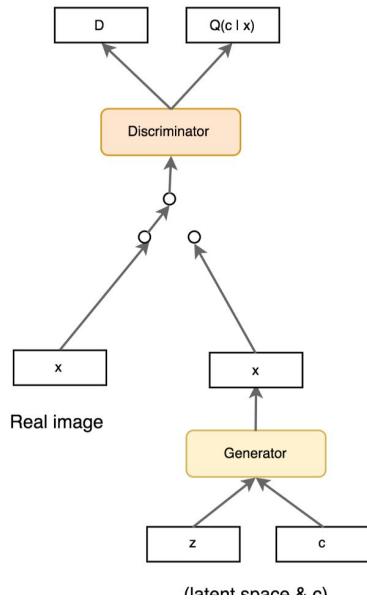
$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

- X and Y are independent, then $I(X; Y) = 0$
- X is completely determined by Y, then $H(Y|X)=0$, $I(X; Y) = H(Y)$

Then I is maximized

CGAN & InfoGAN

- Info GAN (label not given, treated as latent factor):
 - Additional output for the discriminator $Q(c|x)$
 - Mutual information $I(c;x)$ measure how much we know c if we know x . It should be high.
 - The info in the latent code c should not be lost in the generation.



$$\min_G \max_D V_{infoGAN}(D, G) = V_{GAN}(D, G) - \lambda I(c; G(z, c))$$

$$V_{GAN}(D, G) \equiv \mathbb{E}_{x \sim p_{data}} \log D(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z, c)))$$

$$\begin{aligned}
 I(c; G(z, c)) &= H(c) - H(c|G(z, c)) \\
 &= \mathbb{E}_{x \sim G(z, c)} [\mathbb{E}_{c \sim P(c|x)} [\log P(c|x)]] + H(c) \\
 &\geq \mathbb{E}_{x \sim G(z, c)} [\mathbb{E}_{c \sim P(c|x)} [\log Q(c|x)]] + H(c)
 \end{aligned}$$

InfoGAN Results

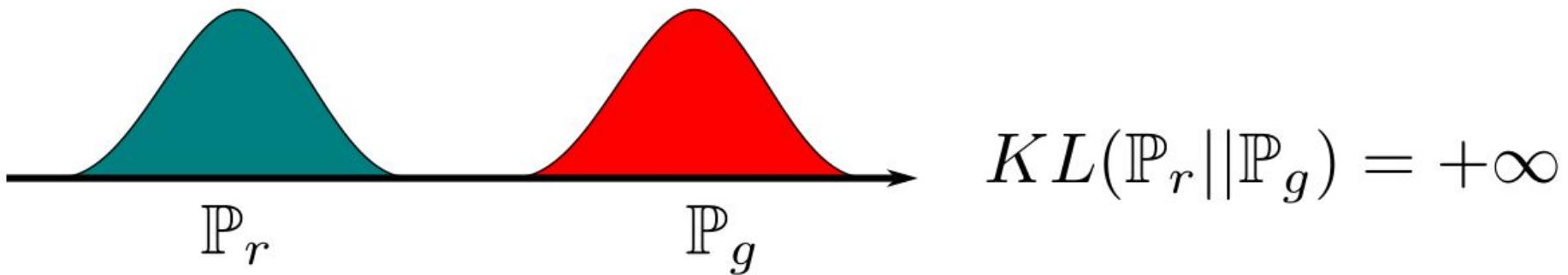


Wasserstein GAN: towards better losses

The *Kullback-Leibler* (KL) divergence

$$KL(\mathbb{P}_r \parallel \mathbb{P}_g) = \int \log \left(\frac{P_r(x)}{P_g(x)} \right) P_r(x) d\mu(x)$$

Problem when the distributions have $P_g(x)=0$ and $P_r(x)>0$:



Wasserstein GAN: towards better losses

The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \parallel \mathbb{P}_m) + KL(\mathbb{P}_g \parallel \mathbb{P}_m)$$

where \mathbb{P}_m is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$.

Minimizing the GAN objective function with an optimal discriminator is equivalent to minimizing the JS-divergence

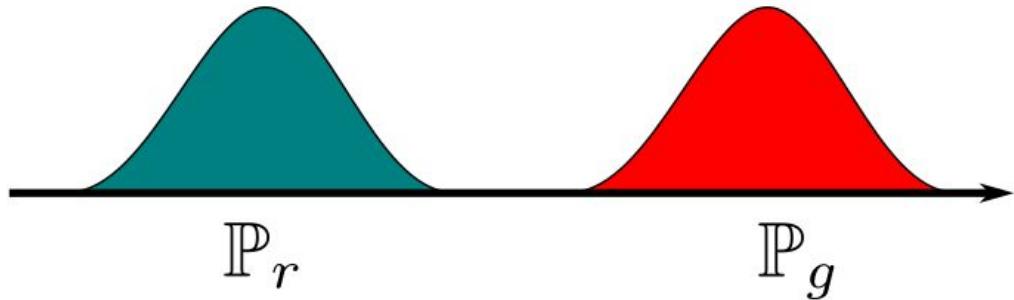
Wasserstein GAN: towards better losses

The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \parallel \mathbb{P}_m) + KL(\mathbb{P}_g \parallel \mathbb{P}_m)$$

where \mathbb{P}_m is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$.

Minimizing the GAN objective function with an optimal discriminator is equivalent to minimizing the JS-divergence



$$JS(\mathbb{P}_r, \mathbb{P}_g) = 2\log(2)$$

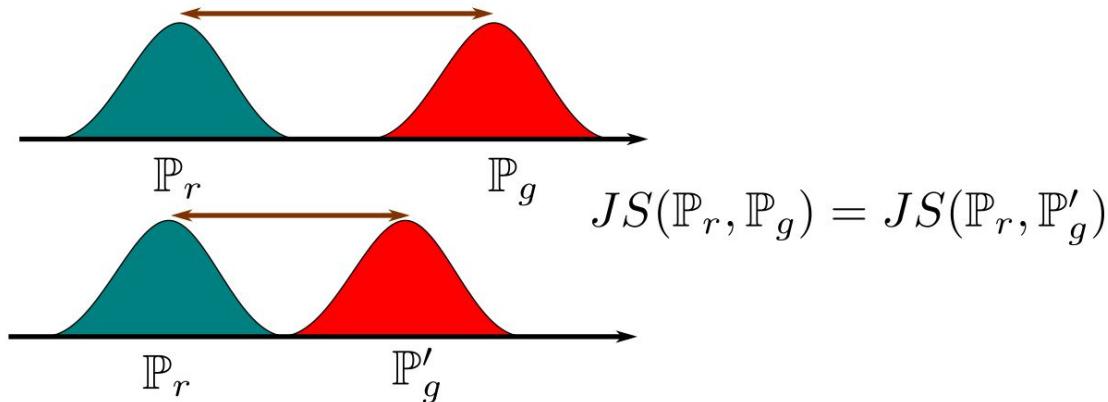
Wasserstein GAN: towards better losses

The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \parallel \mathbb{P}_m) + KL(\mathbb{P}_g \parallel \mathbb{P}_m)$$

where \mathbb{P}_m is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$.

Problem:



The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

Wasserstein GAN: towards better losses

The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

We can show:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} [f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta} [f(x)]$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \rightarrow \mathbb{R}$.

Wasserstein GAN: towards better losses

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

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

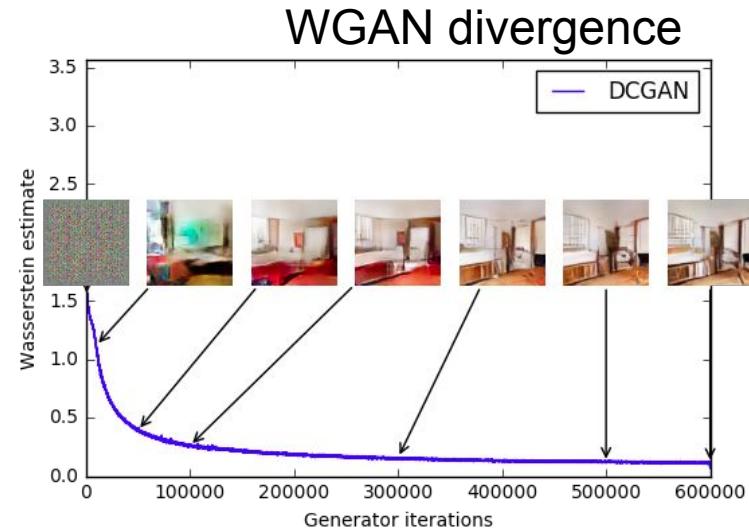
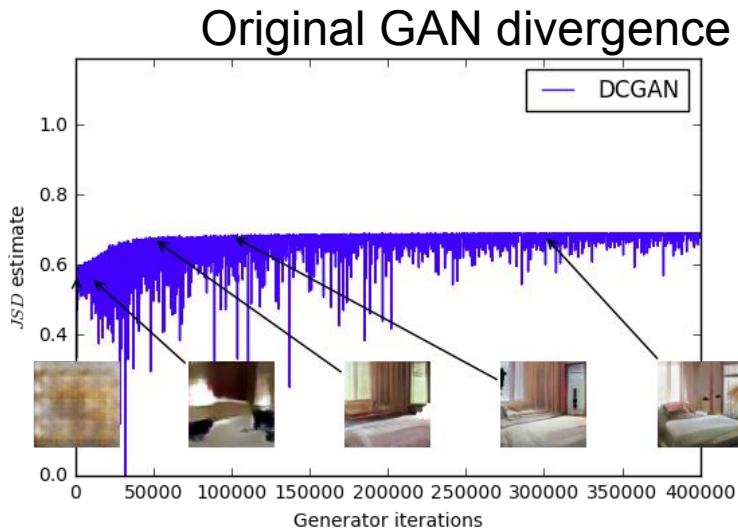
Require: : α , the learning rate. c , the clipping parameter. m , the batch size.
 n_{critic} , the number of iterations of the critic per generator iteration.

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

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

Wasserstein GAN

- Objective function value is more meaningfully related to quality of generator output



Wasserstein GAN: Improved Training (WGAN_GP)

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} [f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta} [f(x)]$$

Weight clipping: $w \leftarrow \text{clip}(w, -c, c)$

Gradient penalty (WGAN-GP):

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

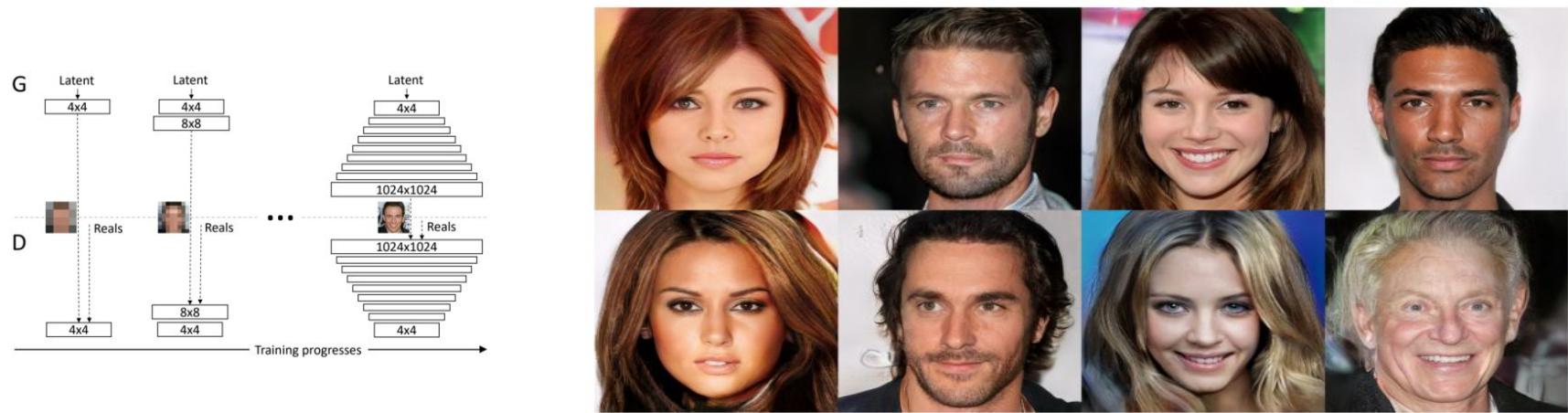
Improved Training of Wasserstein GANs, Gulrajani et.al. Nips 2017

Lost functions

Name	Value Function
GAN	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip_by_value(W_D, -0.01, 0.01)$
WGAN_GP	$L_D^{WGAN_GP} = L_D^{WGAN} + \lambda E[\nabla D(\alpha x - (1 - \alpha G(z))) - 1]^2$ $L_G^{WGAN_GP} = L_G^{WGAN}$
DRAGAN	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[\nabla D(\alpha x - (1 - \alpha x_p)) - 1]^2$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	$L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$ $L_G^{CGAN} = E[\log(D(G(z), c))]$
infoGAN	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c x)] + E[P(class = c G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c G(z))]$
EBGAN	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda (\gamma D_{AE}(x) - D_{AE}(G(z)))$

High Resolution

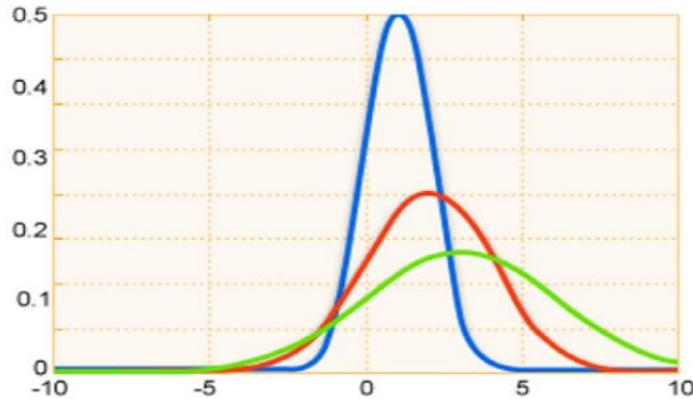
Progressive GAN (2018)



BigGAN (2018)



Instance-normalization



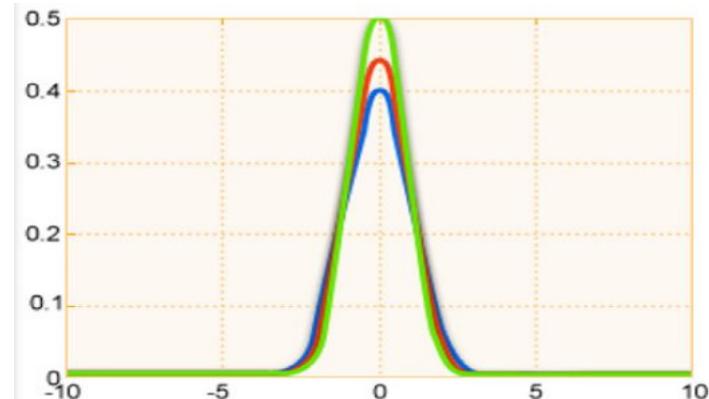
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

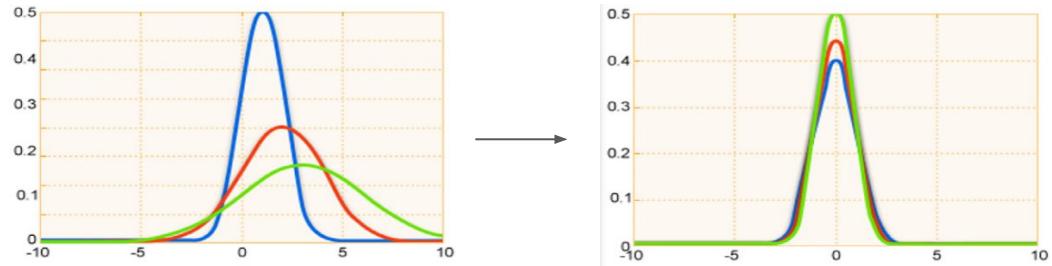


Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
Sergey Ioffe, Christian Szegedy

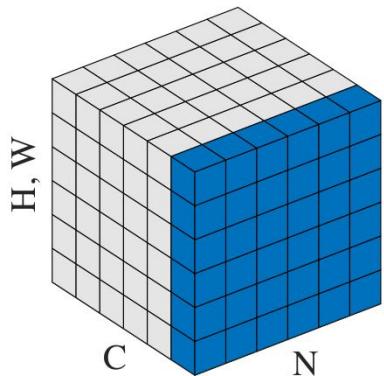
Instance-normalization

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$$

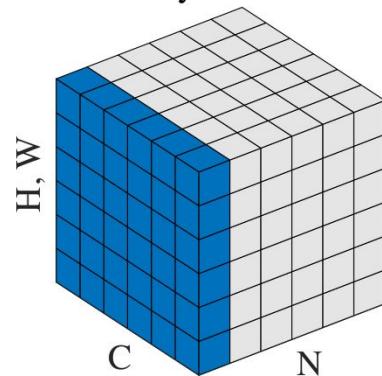
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$$



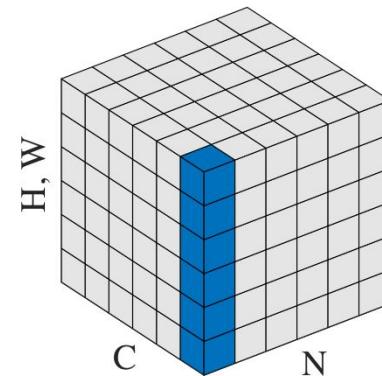
Batch Norm



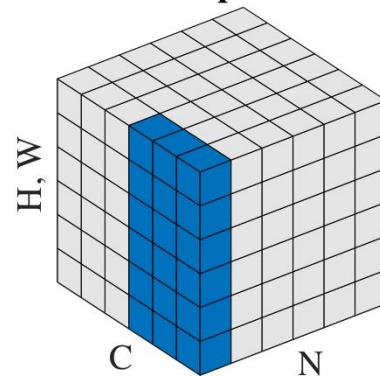
Layer Norm



Instance Norm



Group Norm

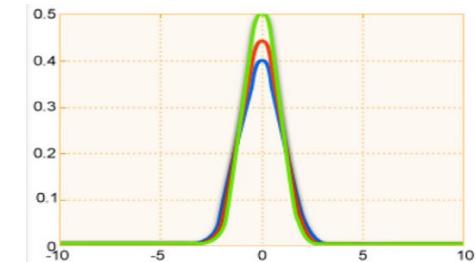
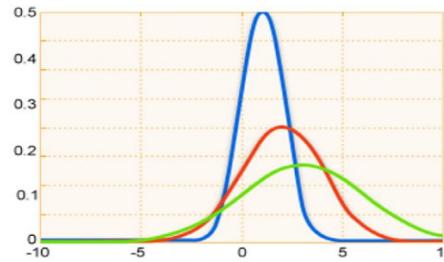


Group Normalization, Yuxin Wu, Kaiming He, ECCV 2018

Why Instance-normalization?

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$$



(a) Content image.

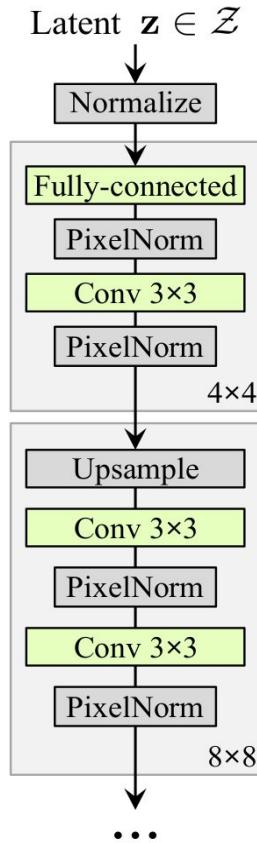


(c) Low contrast content image.

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv$$

The two images are identical after instance normalization.
We obtain style invariant representations.

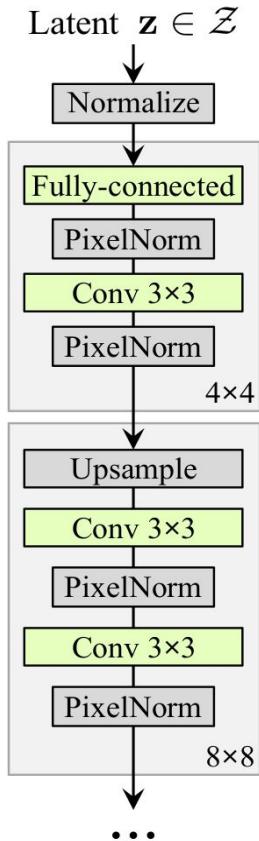
Style-Based GAN



(a) Traditional

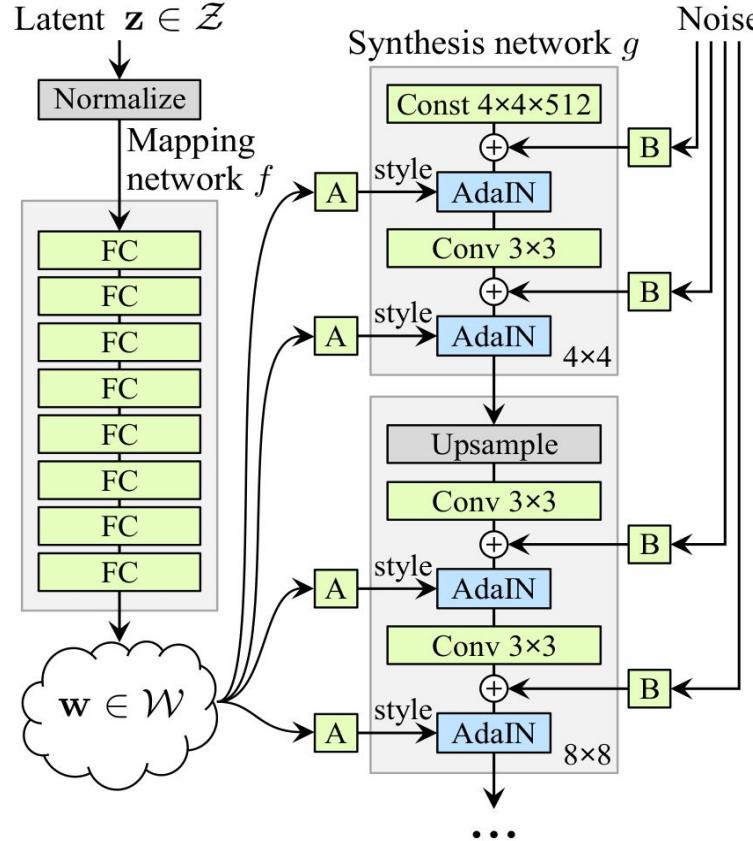
Style-Based GAN

Latent $\mathbf{z} \in \mathcal{Z}$



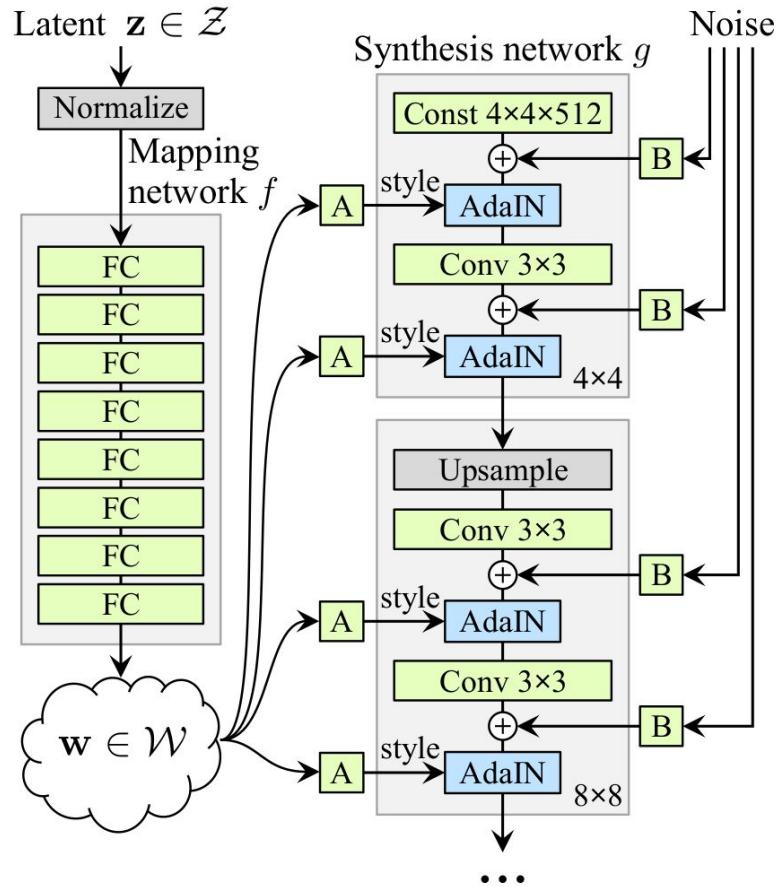
(a) Traditional

Latent $\mathbf{z} \in \mathcal{Z}$



(b) Style-based generator

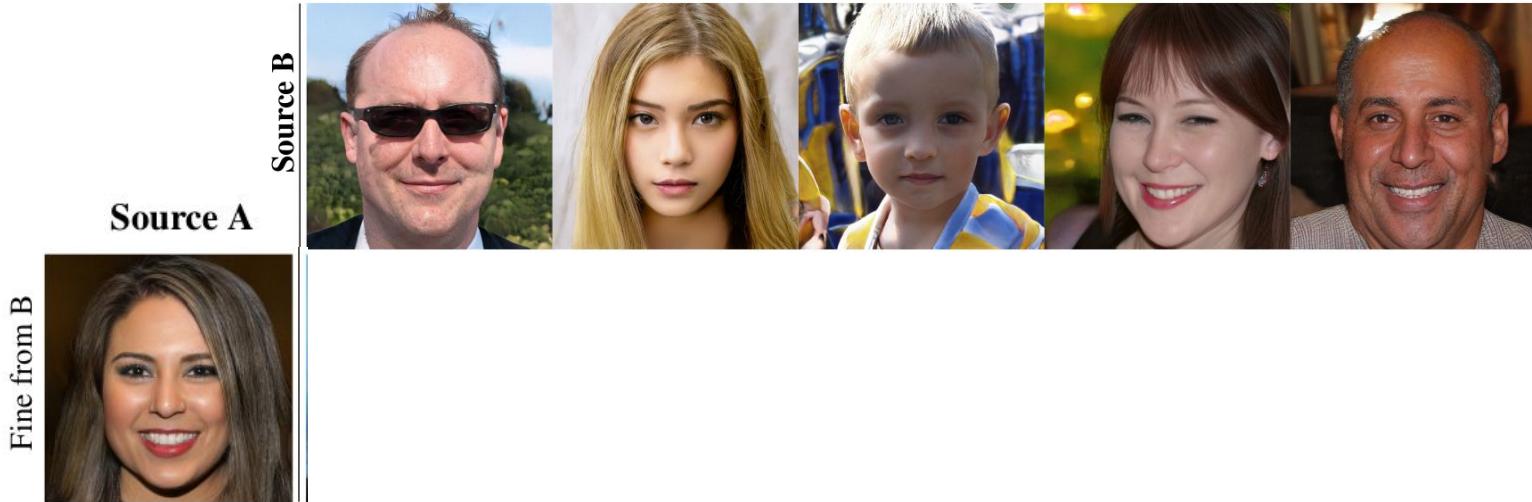
Style-Based GAN



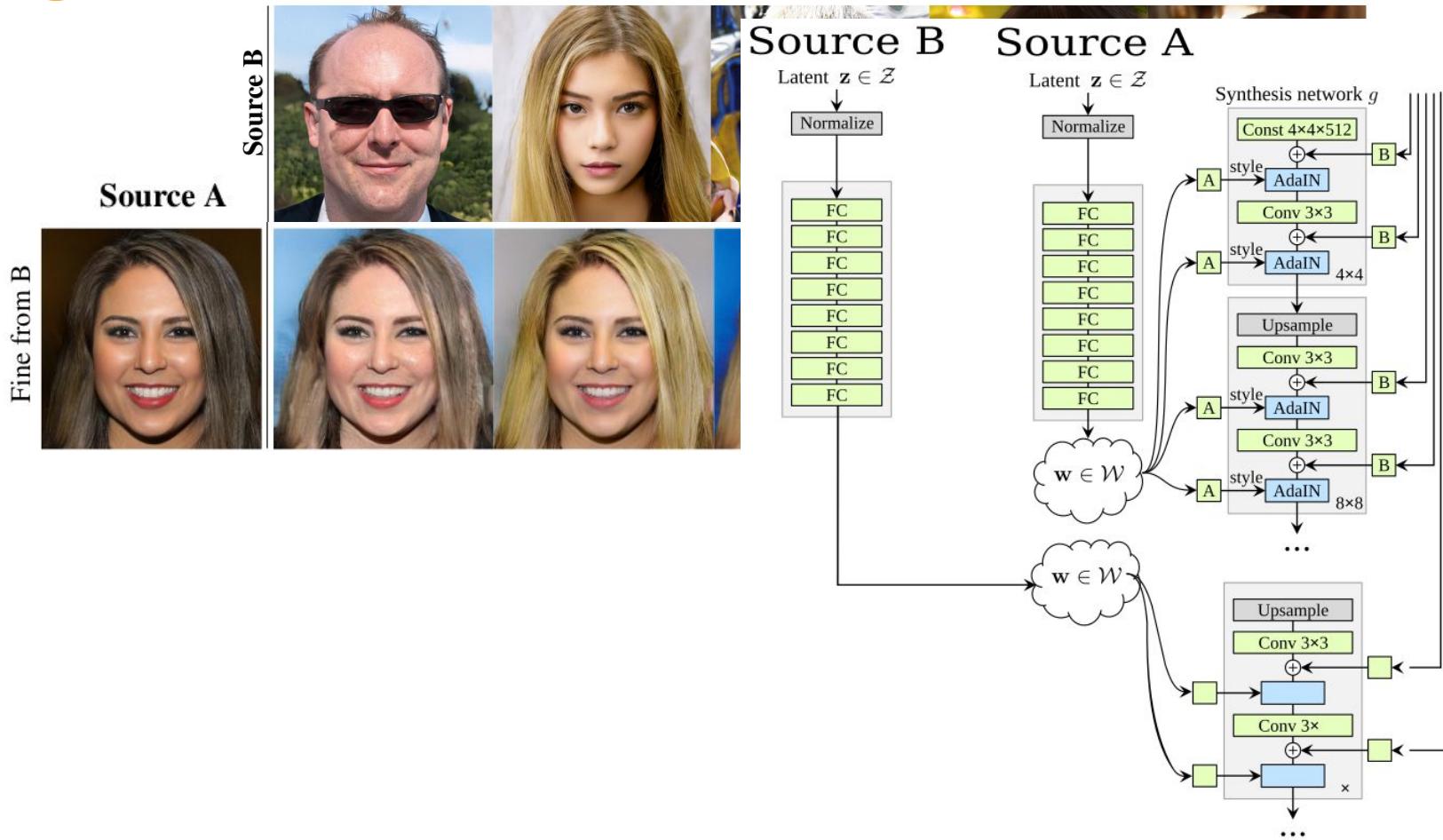
$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

(b) Style-based generator

Style-Based GAN



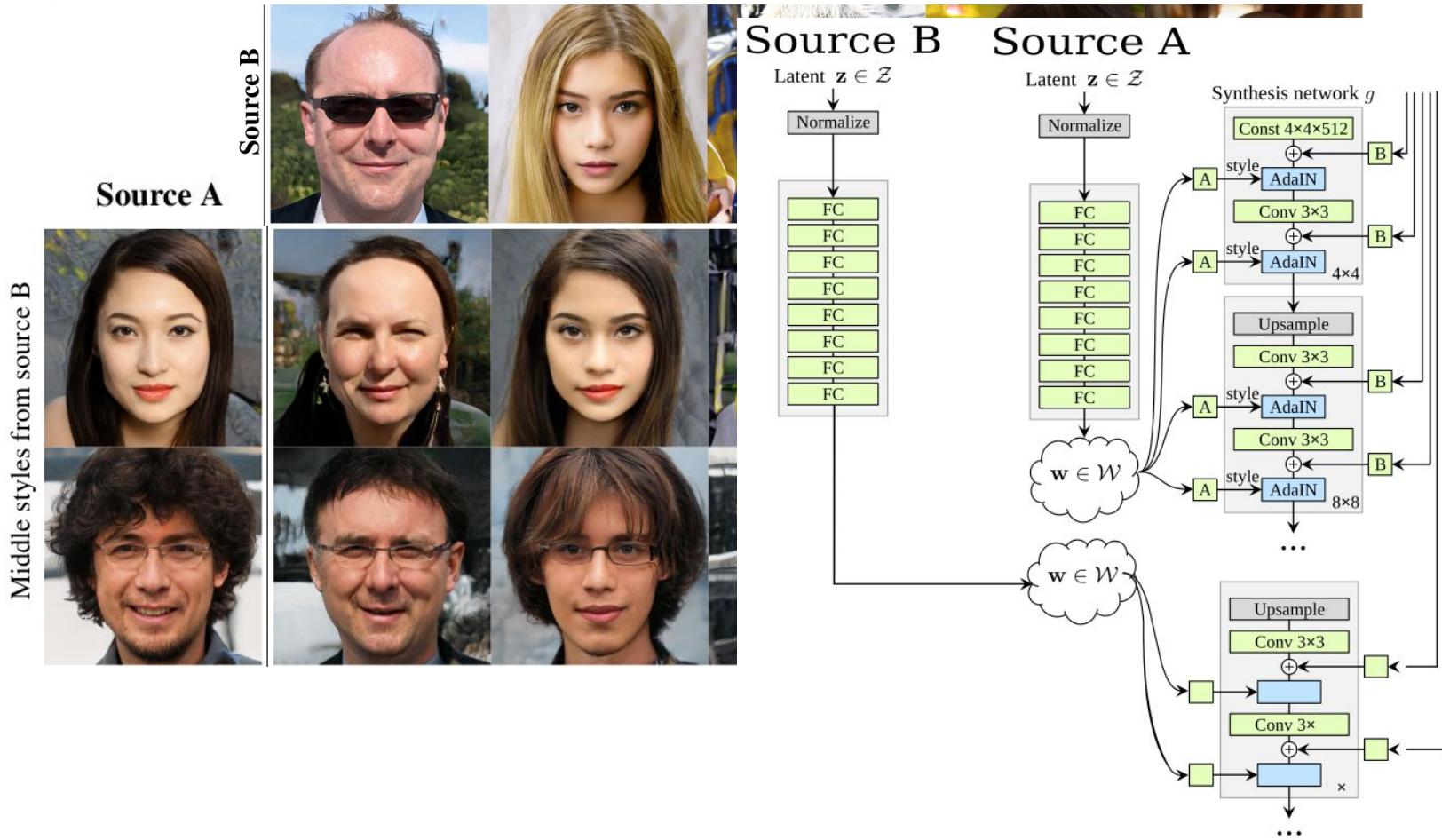
Style-Based GAN



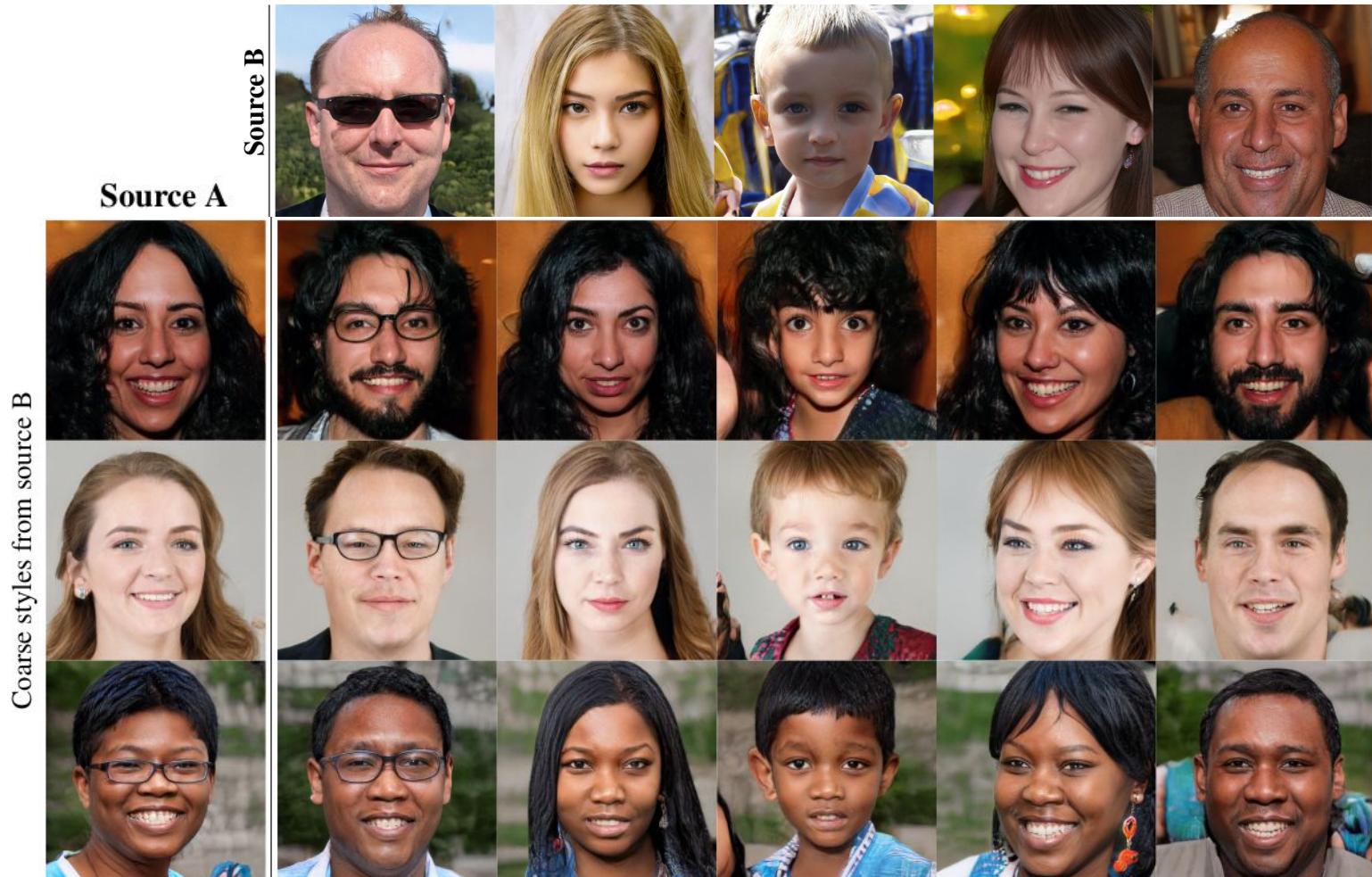
Style-Based GAN



Style-Based GAN



Style-Based GAN



Training GANs: Recap



Training GANs: Recap

Minimax objective function:

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

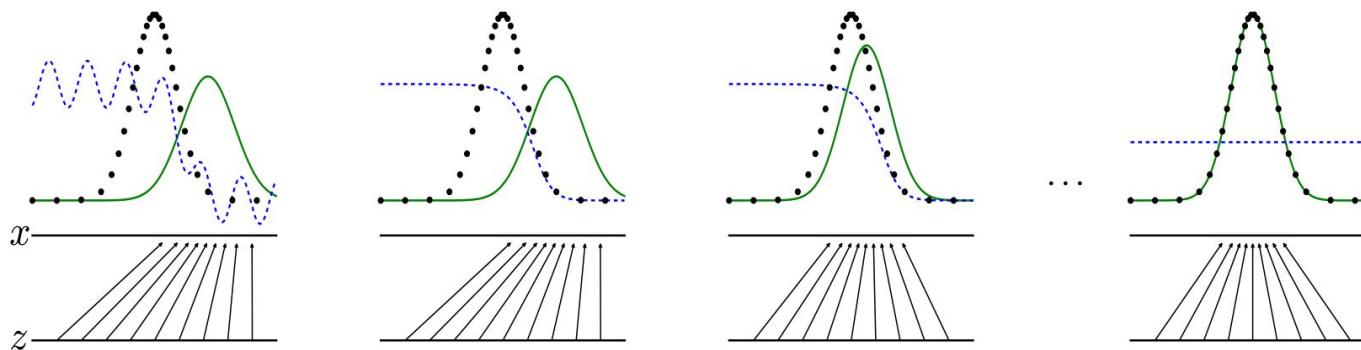
Alternate between:

1. Gradient ascent on discriminator

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

2. Gradient descent on generator

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



Improving GANs: Recap

-
-
-
-
-

Improving GANs: Recap

- Use convolutions
- Use BN
- Many optimization tricks
- Change the loss (e.g. Wasserstein loss)
- Improve the architecture using inference normalization