LG전자 Deep Learning 과정

# Generative Adversarial Networks

Gunhee Kim

Computer Science and Engineering

서울대학교
SEOUL NATIONAL UNIVERSITY

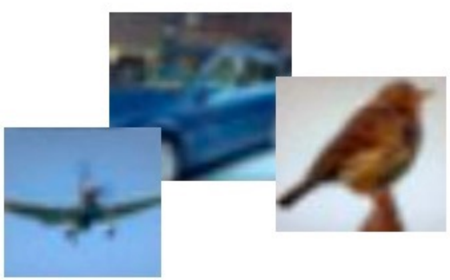
# Outline

- Introduction
- Formulation
- Training Difficulty
- DCGAN

# 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]

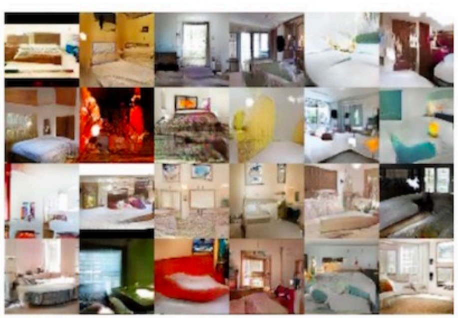
# Generative Models

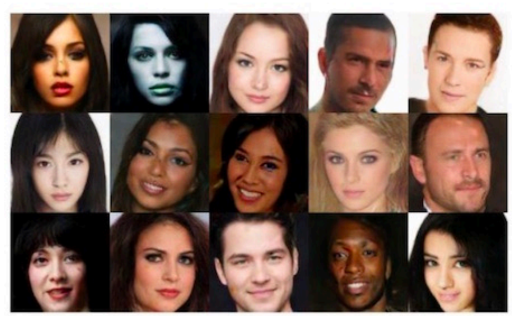
Two approaches

- 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



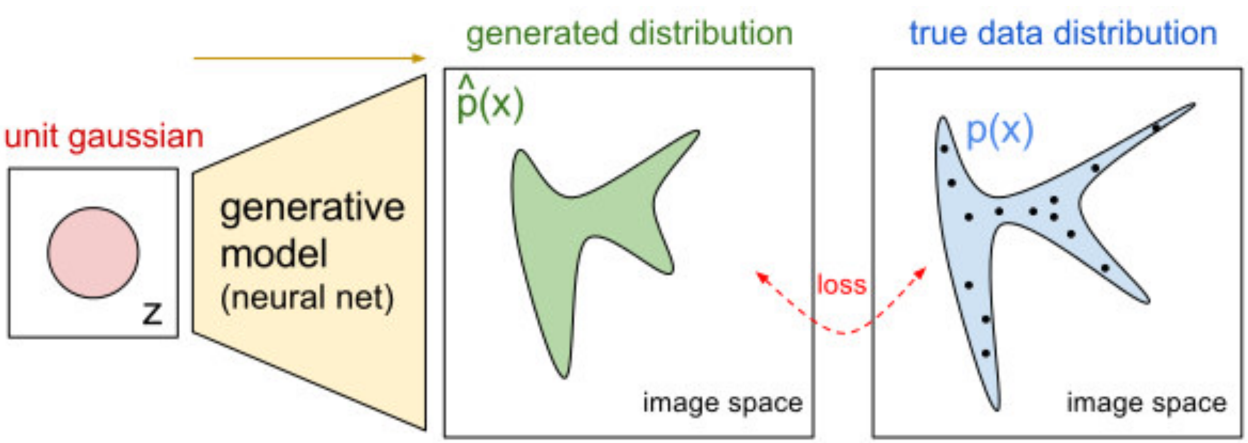


[L.Fei-Fei et al. CS231n]

# Generative Models

## Generative models 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)

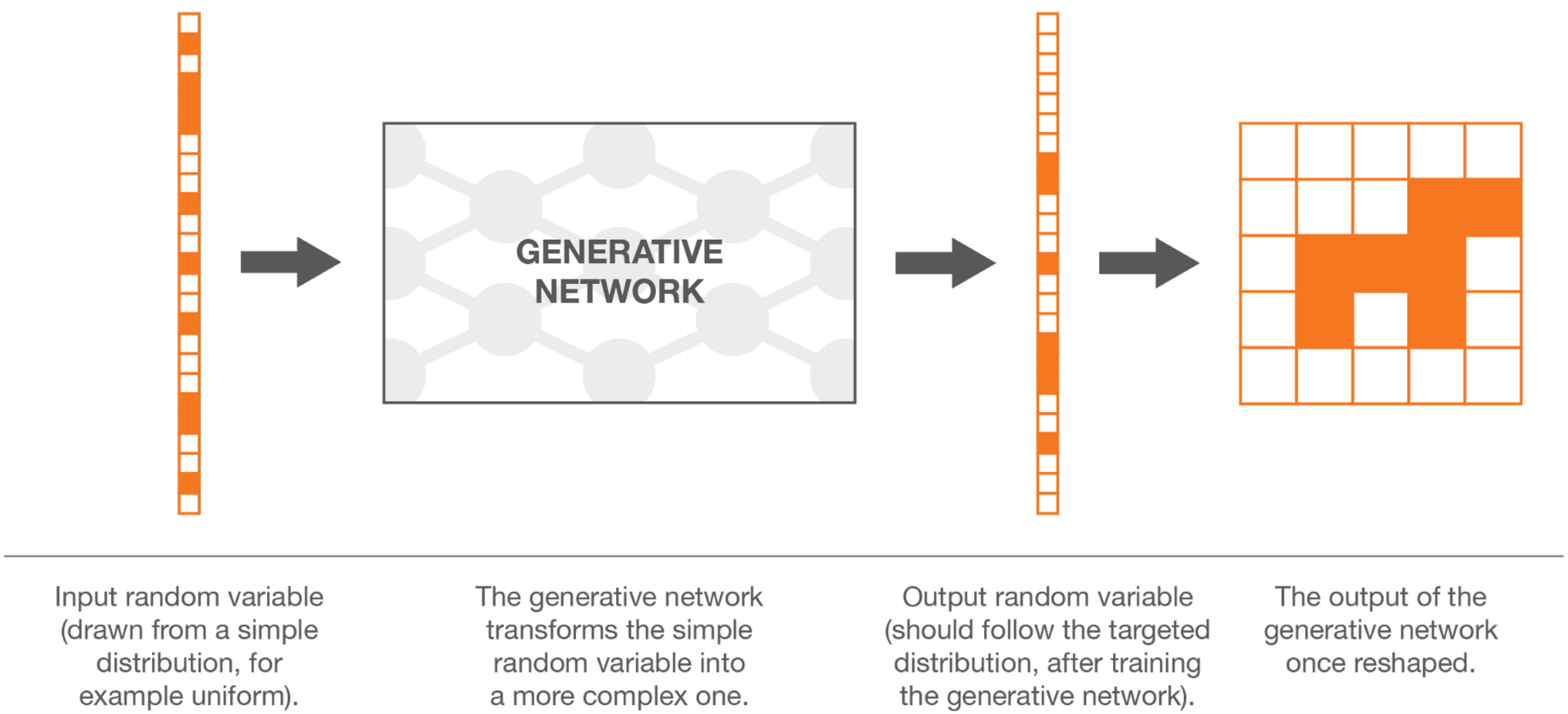



[OpenAI tutorial]

# Generative Models

e.g., generating a dog image with a size of $n \times n$ pixels

- is equivalent to generating a new vector following the dog probability distribution over the $N$-dim vector space ($N = n \times n$)
- challenge: the “dog probability distribution” is a very complex distribution over a very large space

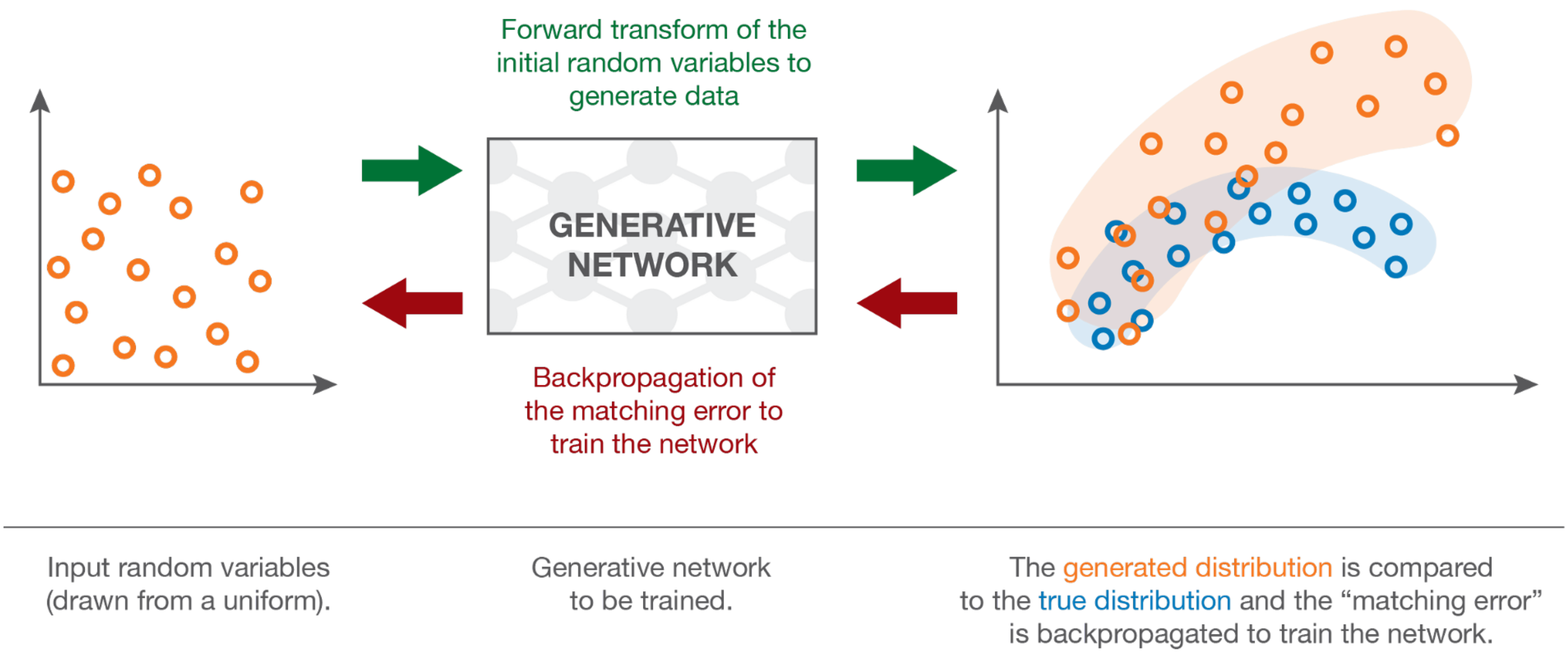



[Joseph Rocca’s tutorial]

# Generative Models

## Generative matching networks

- Take simple random inputs, generate new data, directly compare the distribution of the generated data to that of the true data and backpropagate the matching error for training




[Joseph Rocca's tutorial]

# Generative Adversarial Networks

GANs replace this direct comparison by an indirect one

- Do not directly measure the distance between distributions
- Introduce a discrimination task between true and generated samples

A flexible framework for generative modeling

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



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

# Generative Adversarial Networks

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

- It tries to fool the discriminator

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

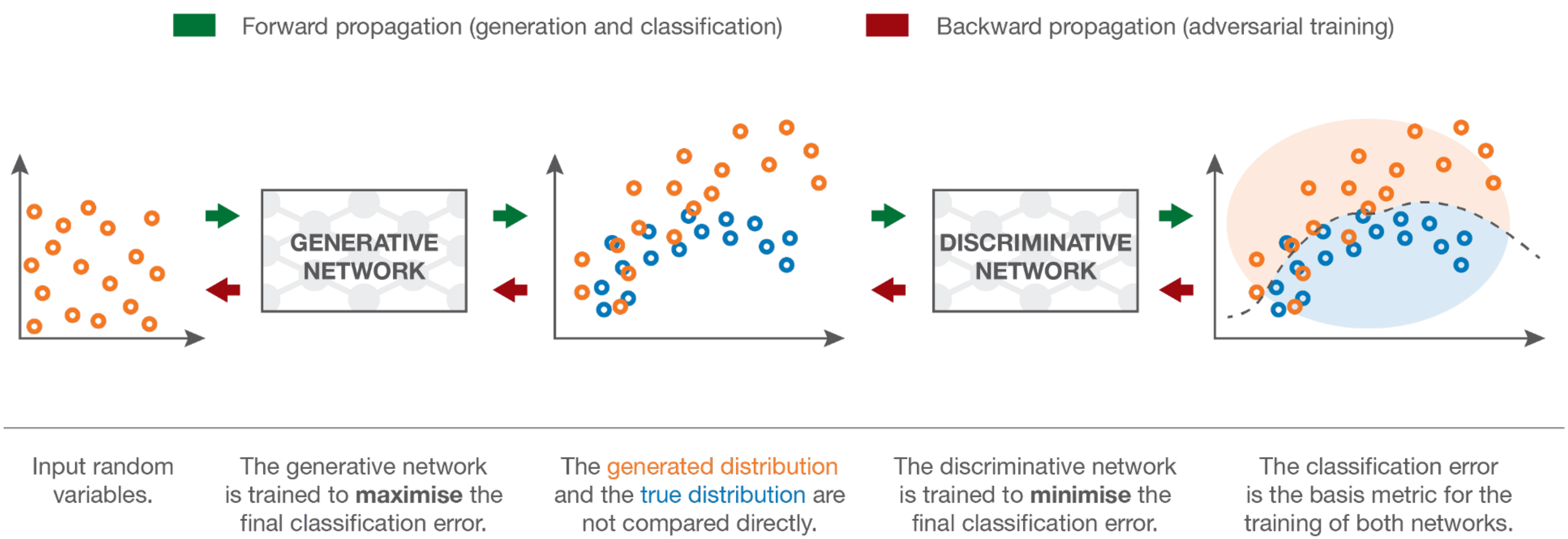
- A binary supervised classifier (real or fake for a given sample)
- It tries to detect faked data accurately as possible

Both G and D are implemented as neural networks

# Generative Adversarial Networks

G tries to maximize the classification error while D minimizes it

- The name of “adversarial networks” is originated
- The classification error becomes the value of minimax two-player zero-sum game



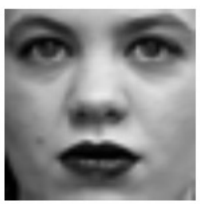

[Joseph Rocca’s tutorial]

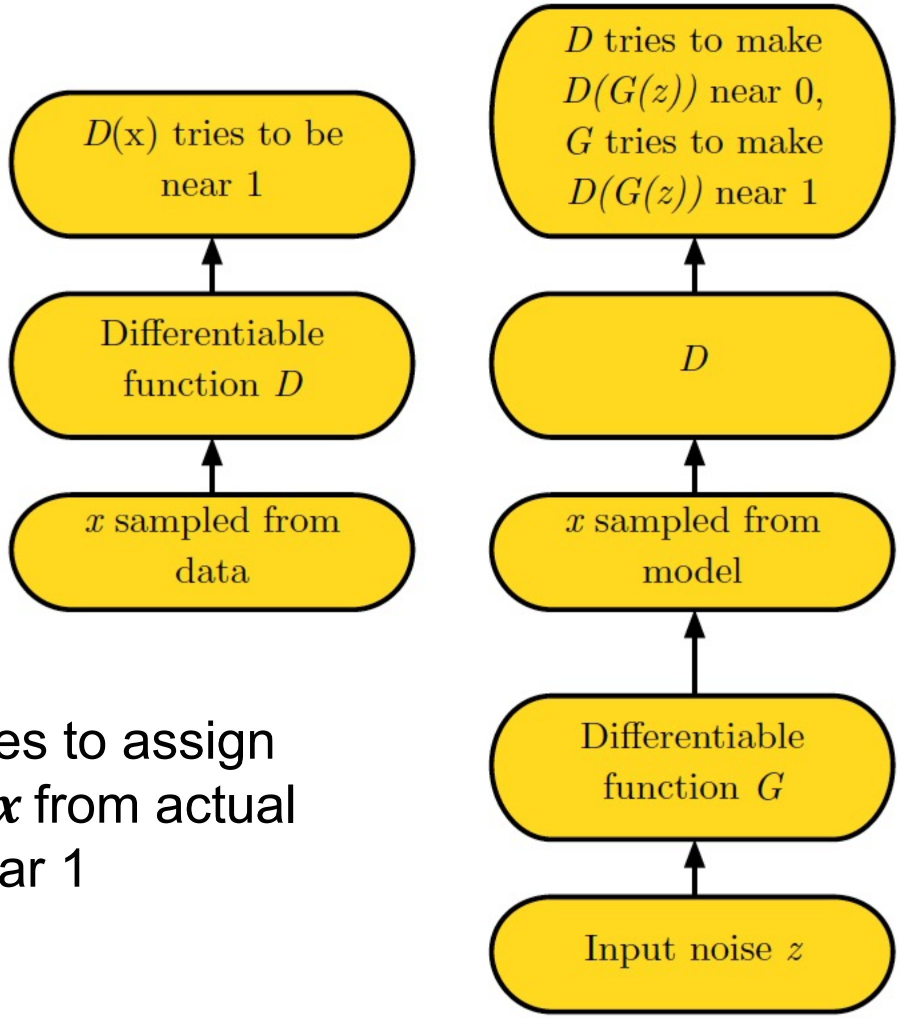
# Outline

- Introduction
- Formulation
- Training Difficulty
- DCGAN

# 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

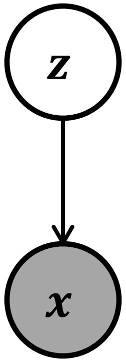
2. $D(\boldsymbol{x})$ tries to assign a sample generated by $G(\boldsymbol{z})$ to near 0

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 $G$ and $D$
- Both should be differentiable w.r.t. their own input and parameters (will use gradient descent)




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

# Cost Functions

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

- The cost for the discriminator is a standard cross-entropy loss

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

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

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

$$\boldsymbol{\theta}^{(G)*} = \arg\min_{\boldsymbol{\theta}^{(G)}} \max_{\boldsymbol{\theta}^{(D)}} V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

# Minimax – Solution to Discriminator

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

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

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

- 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 $p_{mod}(\boldsymbol{x})$ is

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

# Minimax Solution

The training objective $C(G)$ for the optimal D is

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

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

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

$$+ KL(p_{mod}(\boldsymbol{x})|| \frac{p_{data}(\boldsymbol{x}) + p_{mod}(\boldsymbol{x})}{2})$$

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

- Remind that

$$KL(P||Q) = \int_{-\infty}^{\infty} P \ln \frac{P}{Q}\, dx \qquad JSD(P||Q) = \frac{1}{2} KL(P||M) + \frac{1}{2} KL(Q||M), \qquad M = \frac{1}{2}(P+Q)$$

# Minimax Solution

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

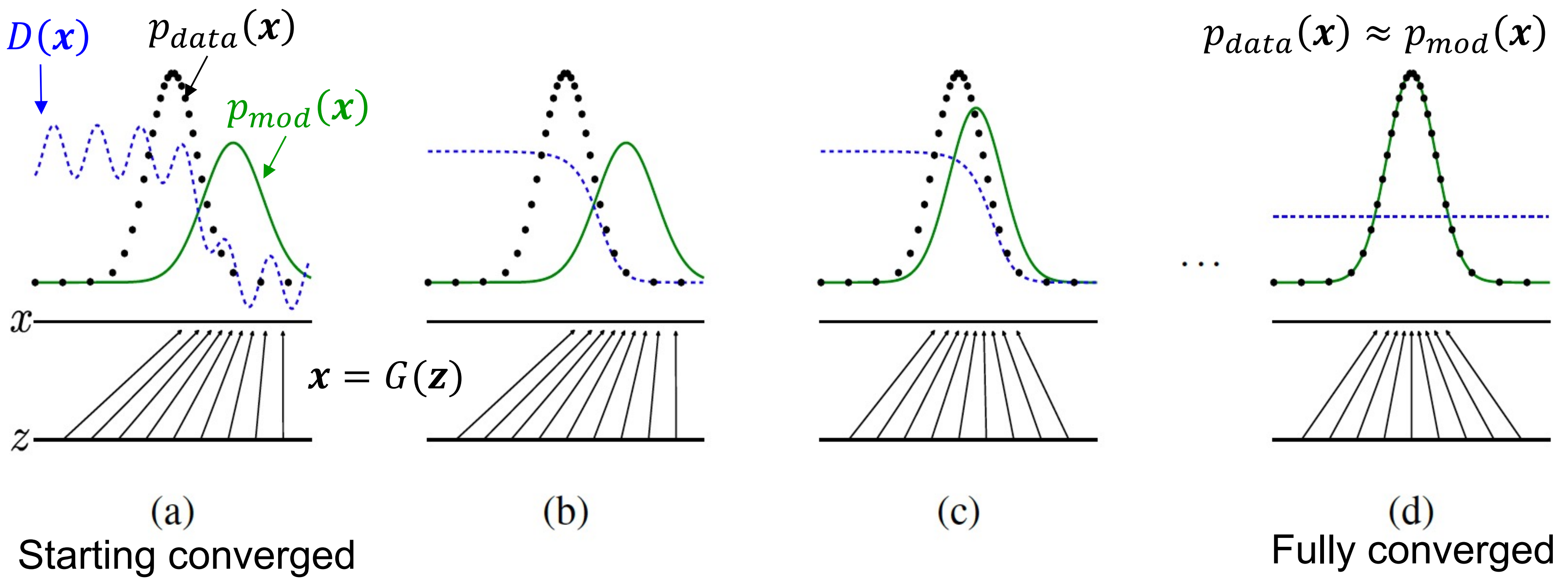
$$C(G) = \min_G V(G, D) = -\log(4) + 2 \cdot JSD(p_{data}(\boldsymbol{x}) || p_{mod}(\boldsymbol{x}))$$

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

- 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 $p_{data} = p_{mod}$,

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

# Learning Process of GAN




- The arrow shows how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ is done by generator
- (a) $D(\boldsymbol{x})$: partially accurate (e.g., high in the left and low in the right)
- (b) $D(\boldsymbol{x})$ is converging to $D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{mod}(\boldsymbol{x})}$
- (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

# Outline

- Introduction
- Formulation
- Training Difficulty
- DCGAN

# Difficulty of GAN Training – Non-Convergence

Deep learning models (in general) involve a single player

$$\min_G L(G)$$

- The player tries minimize its loss using SGD (with backprop)
- SGD has convergence guarantees (under mind conditions)
- Problem: with non-convexity, we might converge to local optima

GANs instead involve two (or more) players

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

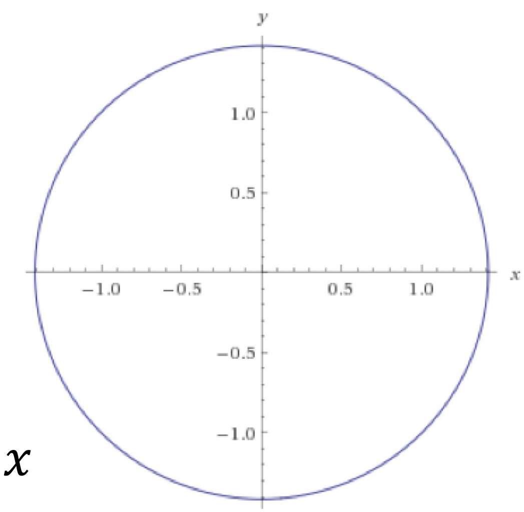
- D tries to maximize its reward, while G minimizes D's reward
- SGD is not designed to find the Nash equilibrium of a game
- Problem: we might not converge to Nash equilibrium at all

# Difficulty of GAN Training – Non-Convergence

1. Non-convergence
   - It is a general problem with games
   - Often the solutions of two-player games are saddle points
   - Nash equilibrium: no agent gains an advantage by switching its strategy

e.g., a simple bilinear game

$$\min_x \max_y \; xy$$




- The Nash equilibrium is $(0,0)$, but the SGD orbits forever around it (blue)
- Player 1 minimizes $V(x, y) = xy$ by controling $x$
- Player 2 minimizes $V(x, y) = -xy$ by controling $y$

# Difficulty of GAN Training – Non-Convergence

e.g., a simple bilinear game

- Can we can find the NE using the gradient descent?
- Player 1 minimizes $V(x, y) = xy$ controling $x$
- Player 2 maximizes $V(x, y)$ controling $y$

$$x \leftarrow x - \alpha \frac{\partial V}{\partial x} = x - \alpha y \qquad y \leftarrow y + \alpha \frac{\partial V}{\partial y} = y + \alpha x$$

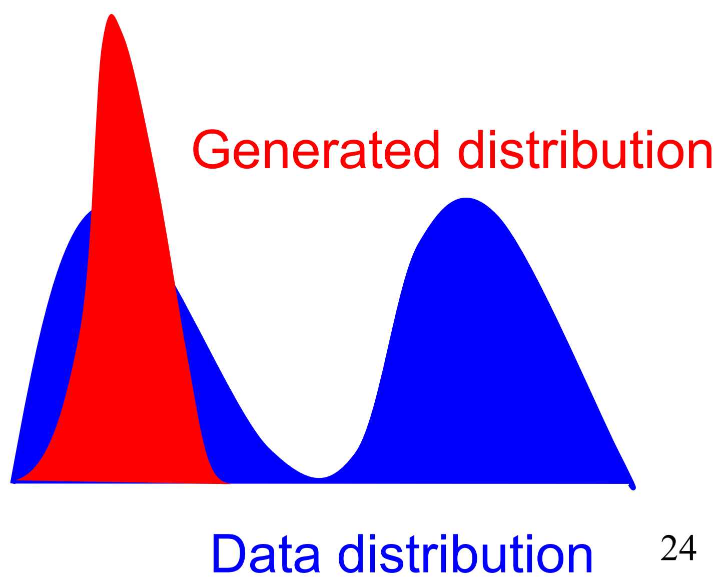
| | | |
|---|---|---|
| $x > 0, y > 0$ | $x$ decreases | $y$ increases |
| $x < 0, y > 0$ | $x$ decreases | $y$ decreases |
| $x < 0, y < 0$ | $x$ increases | $y$ decreases |
| $x > 0, y < 0$ | $x$ increases | $y$ increases |

- GD cannot converge our solution

# Difficulty of GAN Training – Mode Collapse

Generator fails to output diverse samples

- GANs often choose to generate from very few modes (fewer than actual # modes)
- Several different $z$ values to the same output point
- Mode collapse causes low output diversity (e.g., more or less similar generated images)

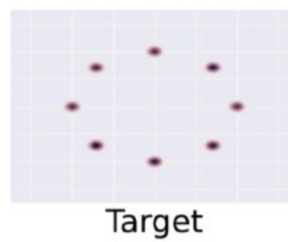


# Difficulty of GAN Training – Mode Collapse

A mode collapse example (a toy 2D mixture of Gaussians)

- $G$ never converges to a fixed distribution
- $G$ only ever produces a single mode at a time, cycling between different modes as $D$ learns to reject each one
- The generator rotates through the modes of the data

Target distribution
$p_{data}$




A heatmap of distributions that GAN learns over time




L. Metz et al. Unrolled generative adversarial networks. ICLR 2017

# Difficulty of GAN Training

3. Unstable dynamics: hard to keep $G$ and $D$ in balance

- Often $D$ overpower $G$ since the task of $D$ (binary classification) is much easier than that of $G$ (data generation
- Too accurate $D$ can vanish the gradient for $G$
- Often use a regularizer to limit the learning progress of $D$

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

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

# Difficulty of GAN Training

4. Quantitative evaluation of $G$ is not easy

- Many desired criteria: fidelity, diversity, …
- There is no clearly justified way to quantitatively score samples
- It can be difficult to estimate the likelihood for GANs (that other generative models)
- Too many metrics have been proposed so far

A. Borji, Pros and Cons of GAN Evaluation Measures, arXiv 2018 (https://arxiv.org/abs/1802.03446)

# Outline

- Introduction
- Formulation
- Training Difficulty
- DCGAN

# Deep Convolutional GAN

DCGAN: use CNNs instead of MLPs

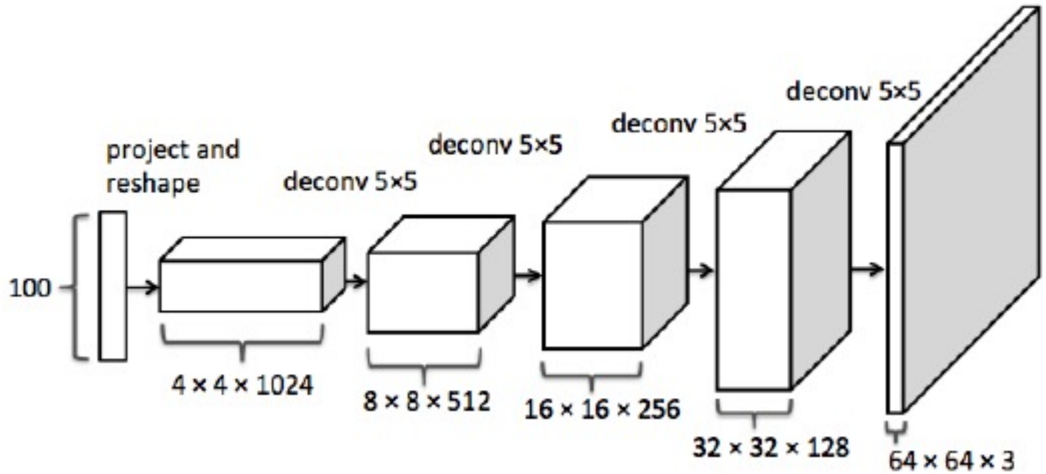
- Generate a realistic image that is different from training images
- $G$ is optimized to fool $D$ (more realistic images)
- $D$ is optimized to not get fooled by $G$


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

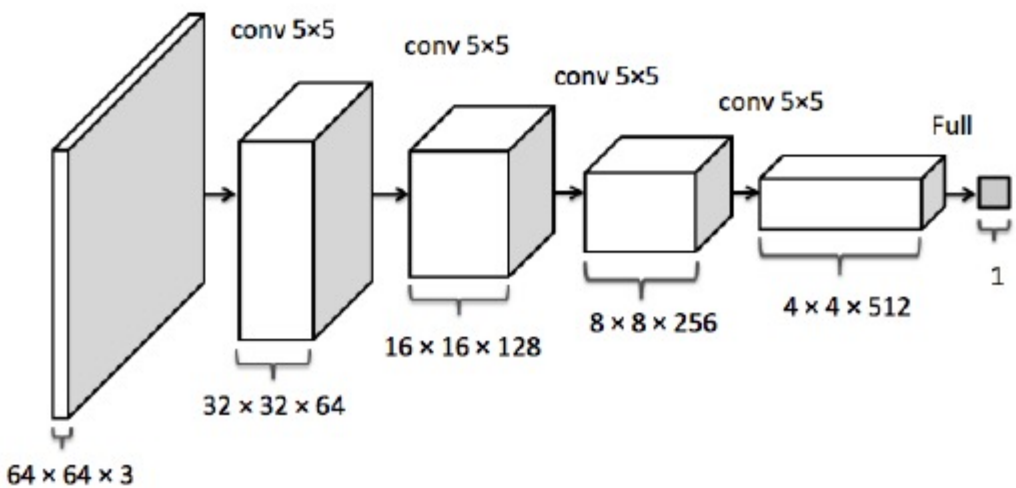
Input: 100-D random numbers




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 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

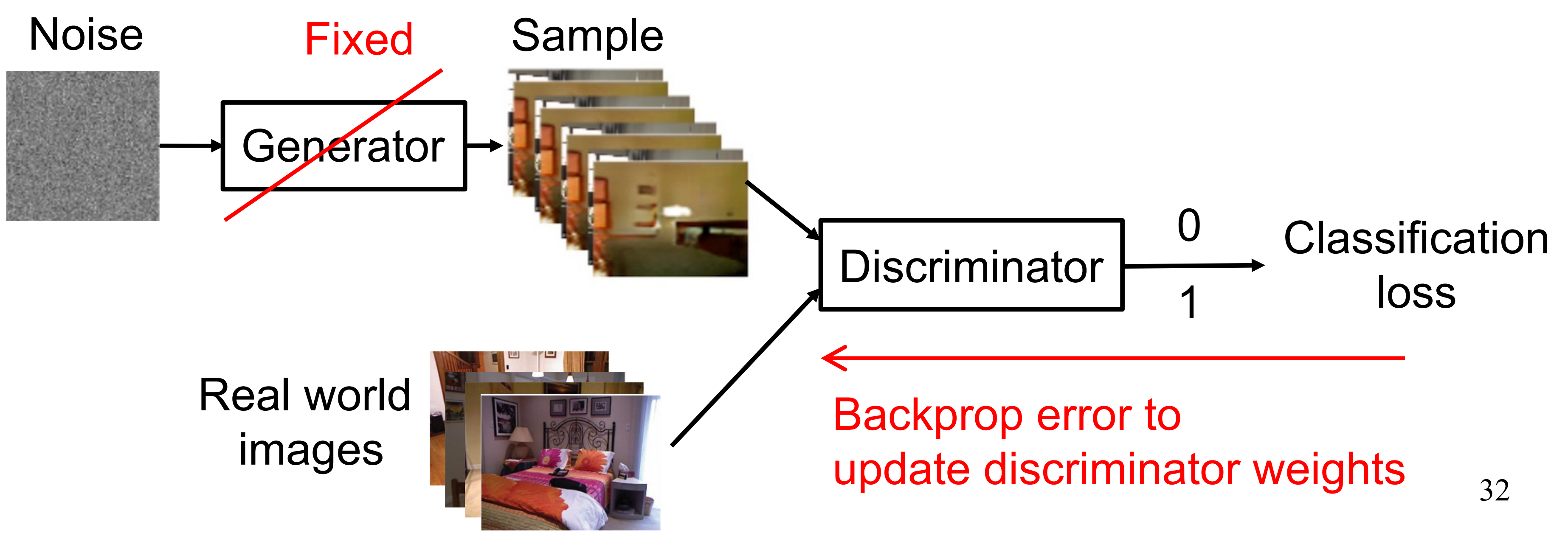
# Training of DCGAN

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



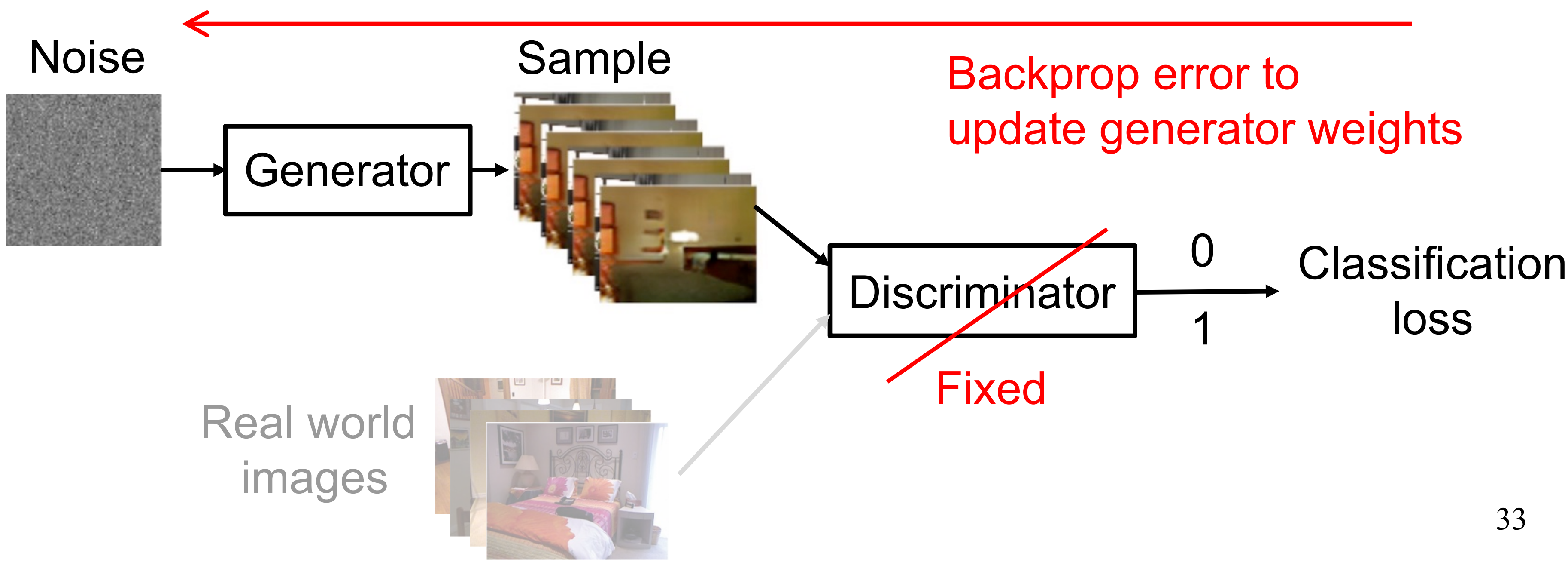
# Training of DCGAN

Alternate between training discriminator and generator

- Note that both $G$ and $D$ should be differentiable

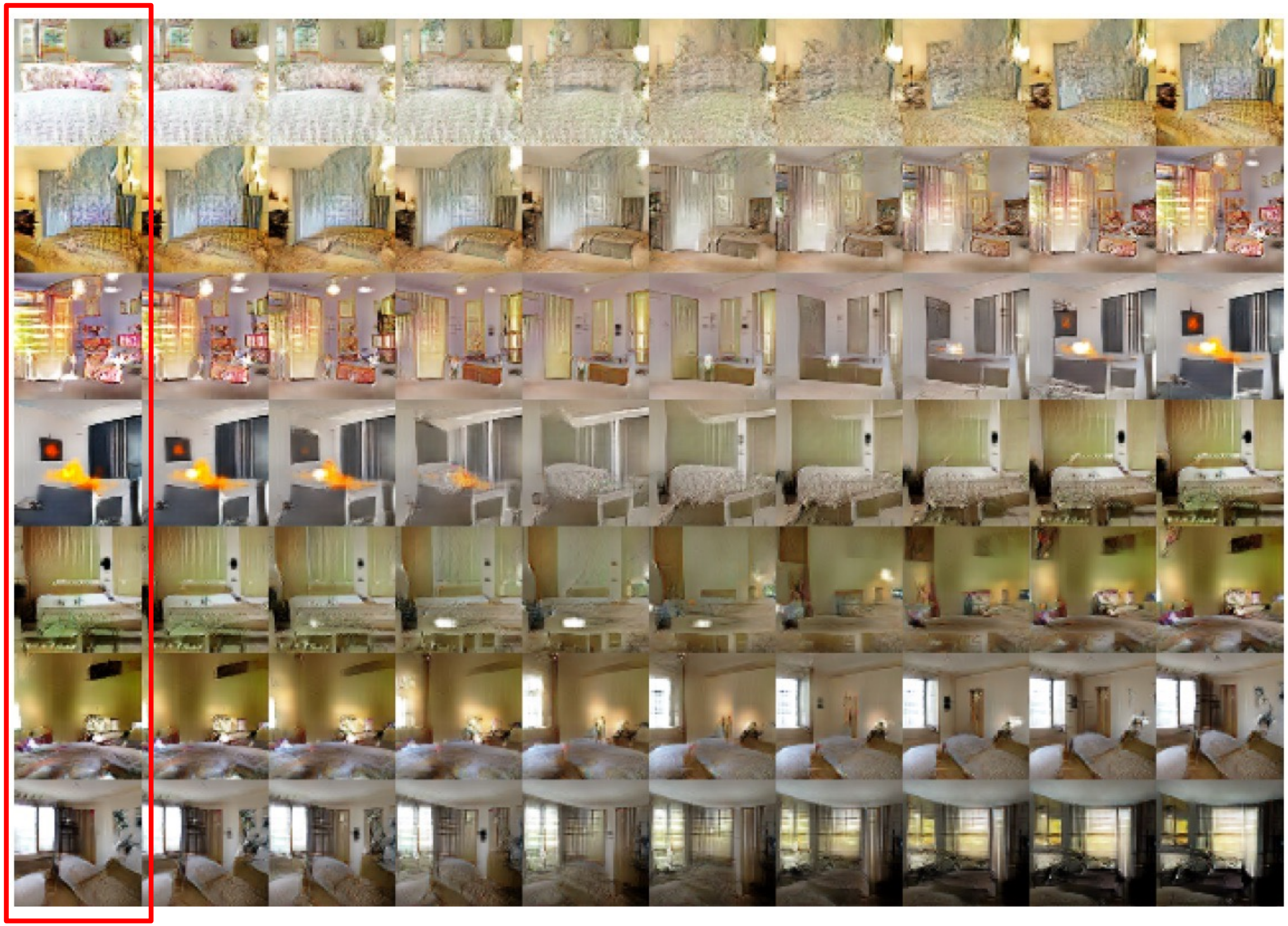
2. Training $G$

- 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

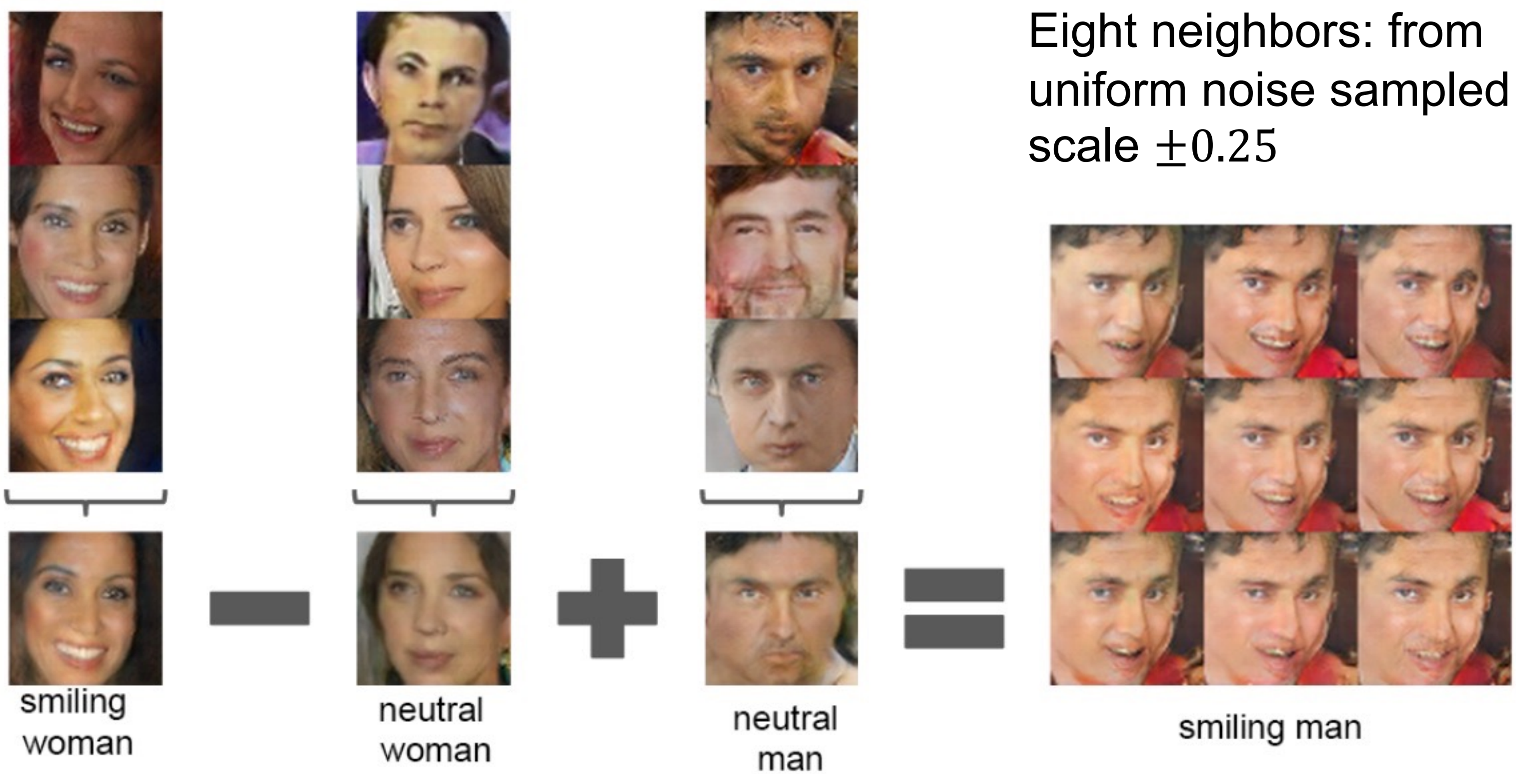


# DCGAN Results – Vector Arithmetic

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

Center: from mean vector
Eight neighbors: from uniform noise sampled with scale $\pm 0.25$



# Conclusion

One of the most successful generative models

Many real-world applications

High-fidelity images

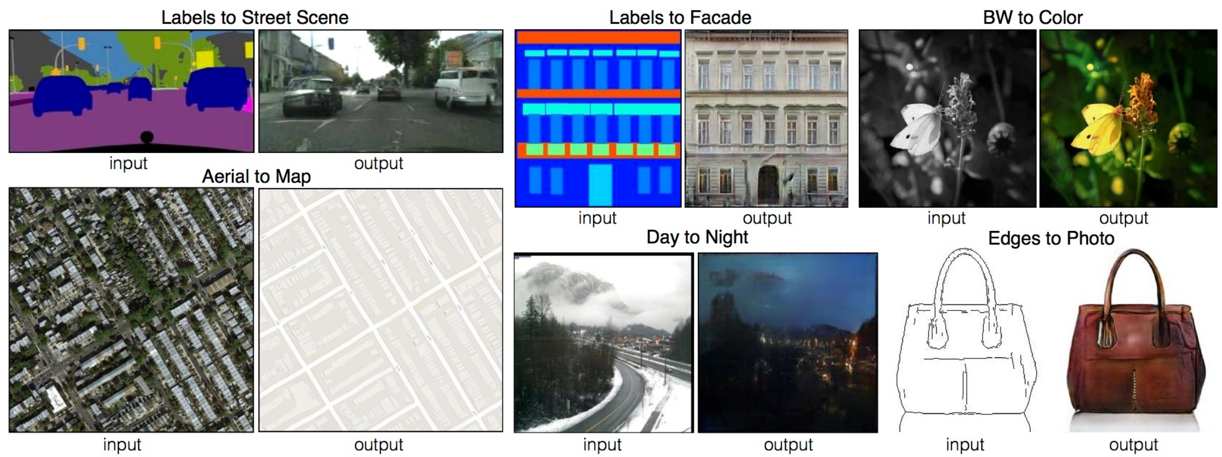



Image-to-image translation

Image inpainting

T. Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019
P. Isola et al., Image-to-Image Translation with Conditional Adversarial Nets, CVPR 2017