Deep Learning

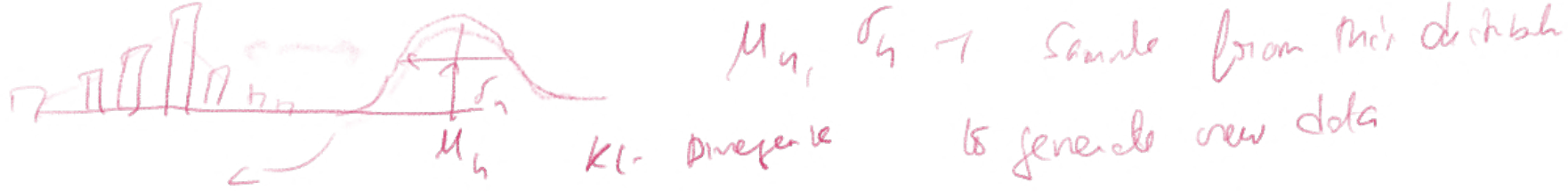
Introduction to Generative Models

Introduction

- Data is in the form of images, videos, text and tables numbers are used for drawing inference - Machine Learning techniques
- Generative models use probabilities to describe/characterize data
- Any data D can be seen as samples from a probability distribution p_{data}
- Generative models try to approximate this distribution given the training data D
- The learned model can then be used for inference tasks

Generative models

- Parametric approximations to data distributions are good generative model- for e.g. if we can model given data as a Gaussian distribution , we can describe the data using the mean and standard deviation
- The distribution can be sampled to generate new data points and the probability of a particular sample x can be easily determined for inference
- The model parameters are learnt by minimising an objective function - a distance metric which measures the distance between distributions



Learning Generative Models

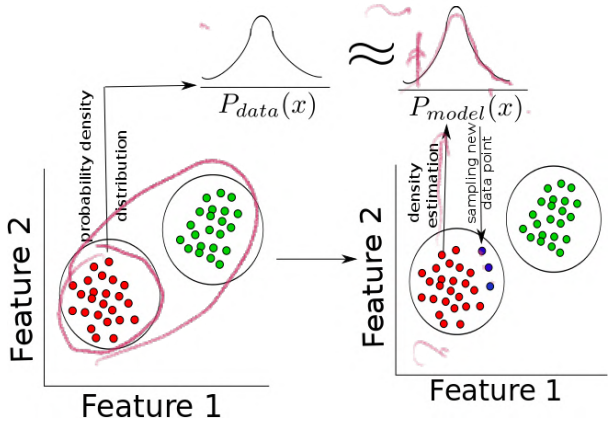
- Distance between the desired distribution and the training Data distribution
- p_θ is the desired distribution parametrised by θ, and θ is estimated such that d(p_θ, p_{data}), where d() is a distance metric between probability distributions
- How to estimate p_{data}, for let's say a table of numbers or for an image set like ImageNet. Challenging?

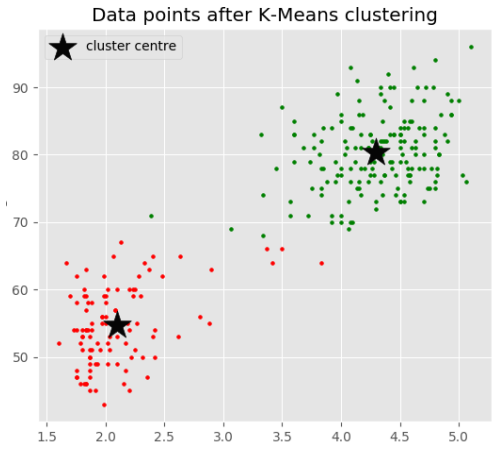
Learning Generative Models

- A typical medical image has about a million (~1e6) voxels or more.
- If voxel values were quantised to take values 0-255, that gives rise to an exponentially large number of images- but amount of data is limited
- However, image data has structure that can be extracted, we can call this structure features - priors
- Better to learn these features automatically rather than defining them for every data set- Deep learning

Inference

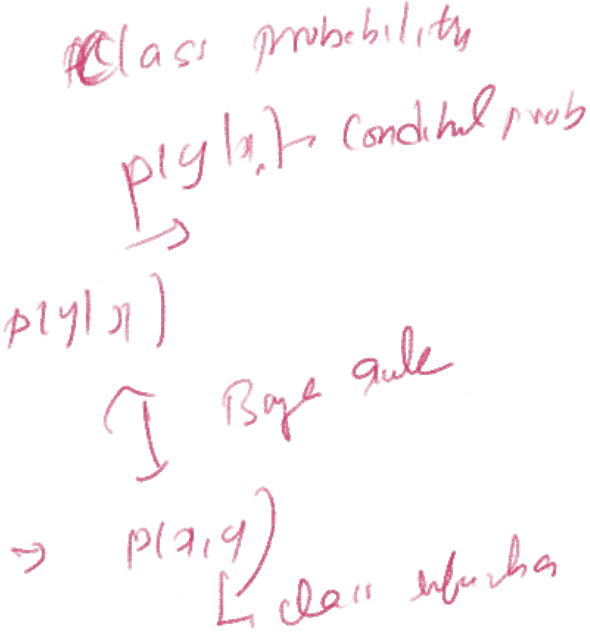
- Typical tasks are
  - Given x what is the probability of the data assigned by the model $p_\theta(x)$?
  - How to generate new data/ synthetic data by sampling $p_\theta(x)$?
  - Can we learn features without the labels?





Discriminative vs Generative

- Discriminative models
  - Estimate class probabilities given the features
  - Model the boundary between different classes
- Generative models
  - Model probability distribution of the data corresponding to individual classes
  - Of course they generate new data



Applications

- Super resolution
- Colorisation of grey scale images
- Predicting occlusions in images
- Cartoon to image conversion, image modifications
- Generative models of speech, text to speech

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, by Ledig et al.

Why Generative Models? (Application)

1. Generates new images, using image to image translation

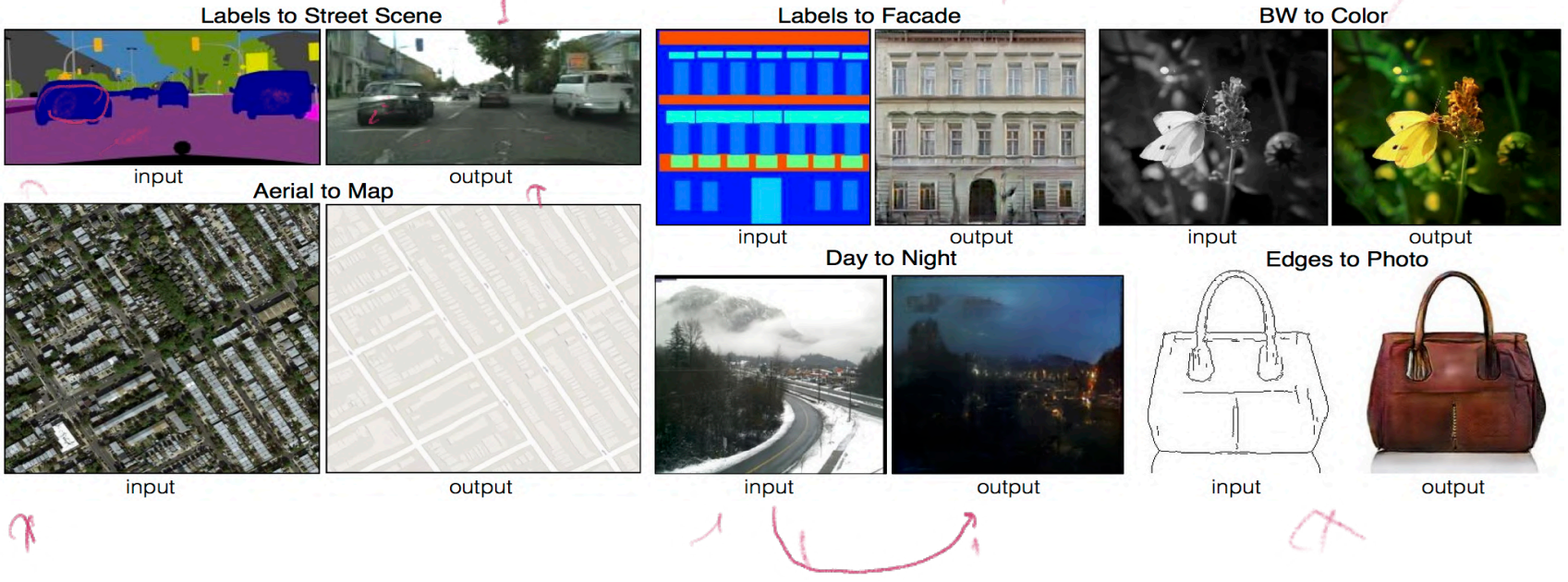


Image-to-Image Translation with Conditional Adversarial Networks, by Isola et al.

Applications

- Generative models of faces - various models over the last 5 years have seen improved results

LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS

Andrew Brock[*†] Heriot-Watt University ajb5@hw.ac.uk
Jeff Donahue[†] DeepMind jeffdonahue@google.com
Karen Simonyan[†] DeepMind simonyan@google.com

[StyleGAN, Karras, Laine, Aila, 2018]

Image Translation

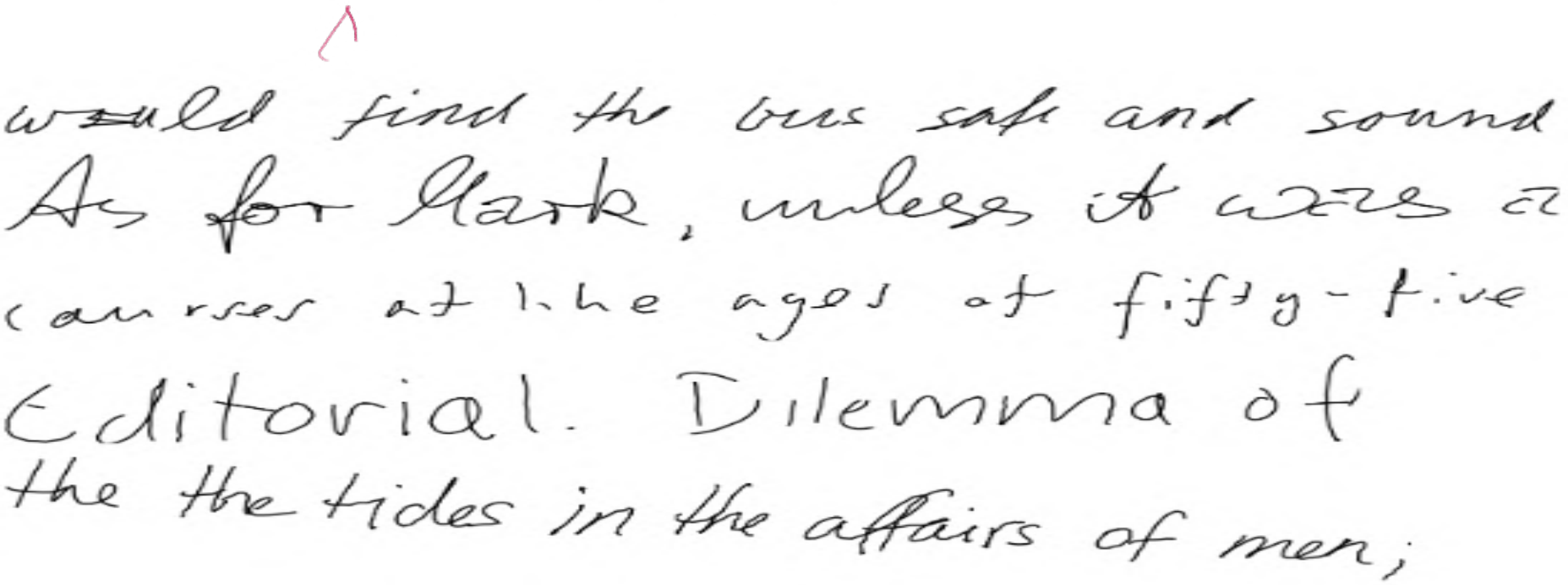
[CycleGAN: Zhu, Park, Isola & Efros, 2017]

https://github.com/junyanz/CycleGAN

2. Generating speech from text

WaveNet: A generative model for raw audio, by Van Den Oord et al.

3. Generating Sequences → (ChatGPT, BERT, LLong BARD)



Generating Sequences With Recurrent Neural Networks, by Alex Graves

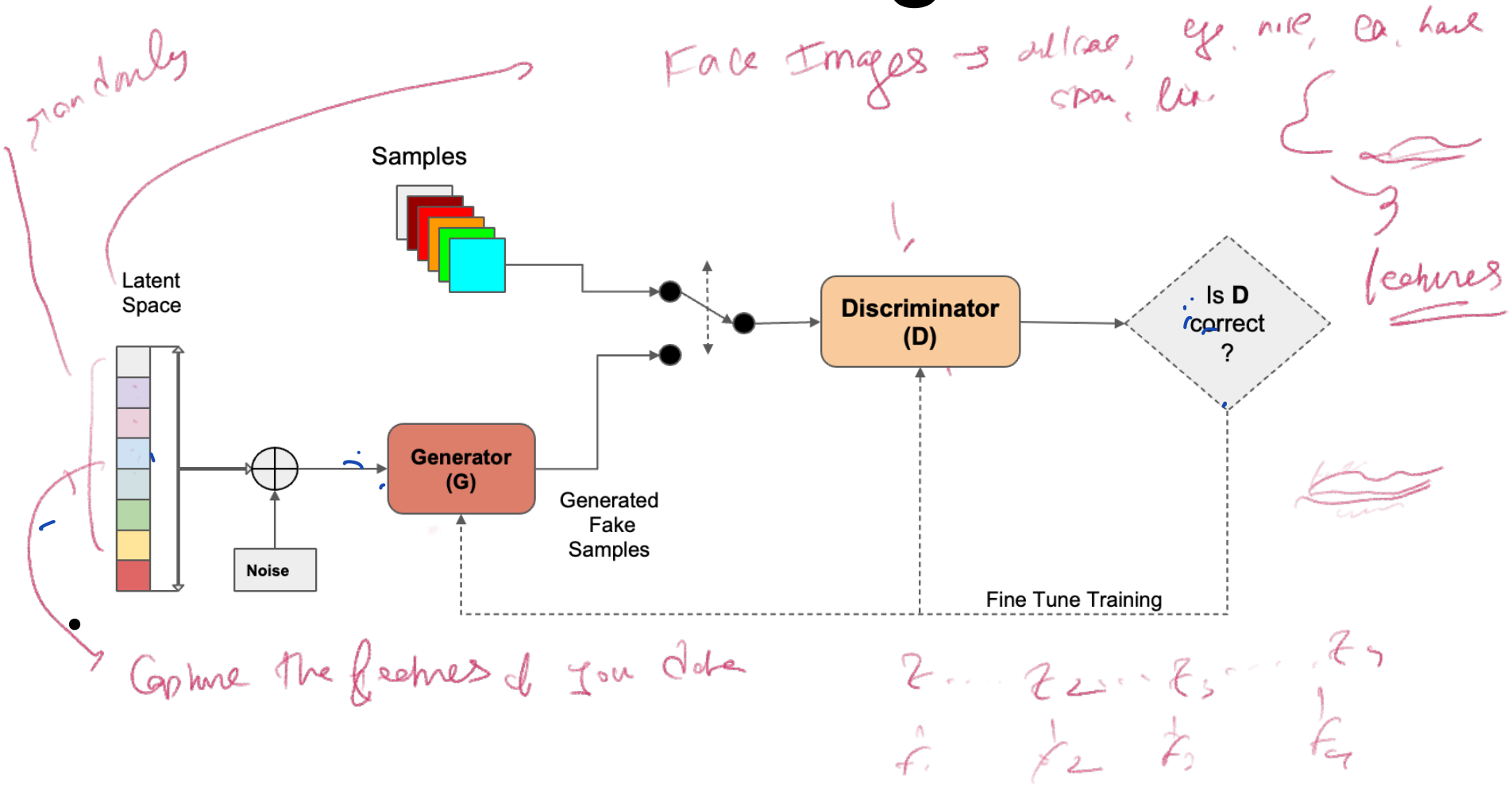
Types of Generative Models

- Explicit density model: explicitly defines and solve for P model (X)
  - Tractable density models/Fully visible belief networks
    - Example: Neural autoregressive density estimator (NADE), Masked autoencoder density estimator (MADE), PixelRNN and PixelCNN
- Approximate density models
  - Variational (Variational autoencoder)
  - Markov chain (Boltzmann Machine)
- Implicit density model: samples from P model (X) without explicitly defining it
  - Markov chain models (GSN)
  - Direct models (GAN)

Types of Generative Models

- Implicit Models , eg- Generative Adversarial Network
- Latent Variabel Models, eg - Variational Autoencoders

Understanding GANs



Understanding GANs

Minmax

Zero sum game formulation

- **Discriminator cost:**

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim P_{data}} log(D(x)) - \frac{1}{2}\mathbb{E}_z log(1 - D(G(z)))$$

- **Generator cost:**

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

- **Value function:**

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

- **Minmax Game:**

$$\min_G \max_D V(\theta^{(D)}, \theta^{(G)}) = \mathbb{E}_{x\sim P_{data}} logD(x) + \mathbb{E}_{z\sim P_z} log(1 - D(G(z)))$$

Discriminator, Generator
output → D(x) G(z) ← latent variable
err → Img
x → Image

$$\min_G \max_D V(\theta^{(D)}, \theta^{(G)}) = \mathbb{E}_{x \sim P_{data}} logD(x) + \mathbb{E}_{z \sim P_z} log(1 - D(G(z)))$$

$\theta^{(D)}$ → Parameter / weights of Discriminator

$\theta^{(G)}$ → Parameter / weights of Generator

$$\min_G \max_D V(\theta^{(D)}, \theta^{(G)}) = \mathbb{E}_{x \sim P_{data}} logD(x) + \mathbb{E}_{z \sim P_z} log(1 - D(G(z)))$$

x is real → D(x) → 1 → log D(x) → 0
G(z) → D(G(z)) → 0 → log(1 − D(G(z))) → 0
} D classifies correctly

x is real → D(x) → 0 → log D(x) → −∞
G(z) → D(G(z)) → 1 → log(1 − D(G(z))) → −∞

Generator ↓

Loss fn / err

$$\min_G \max_D V(\theta^{(D)}, \theta^{(G)}) = \mathbb{E}_{x \sim P_{data}} logD(x) + \mathbb{E}_{z \sim P_z} log(1 - D(G(z)))$$

G(z) → fake image
1) D(G(z)) → 0 → log(1 − 0) → 0
2) D(G(z)) → 1 → log 0 → −∞

V ~ small number

$\theta^{(G)} \sim \theta^{(G)} - \eta \nabla$

→ $\mathbb{E}_{z \sim P_z} \log(D(G(z)))$

D(G(z)) ~ 0 → −∞
D(G(z)) ~ 1 → 0

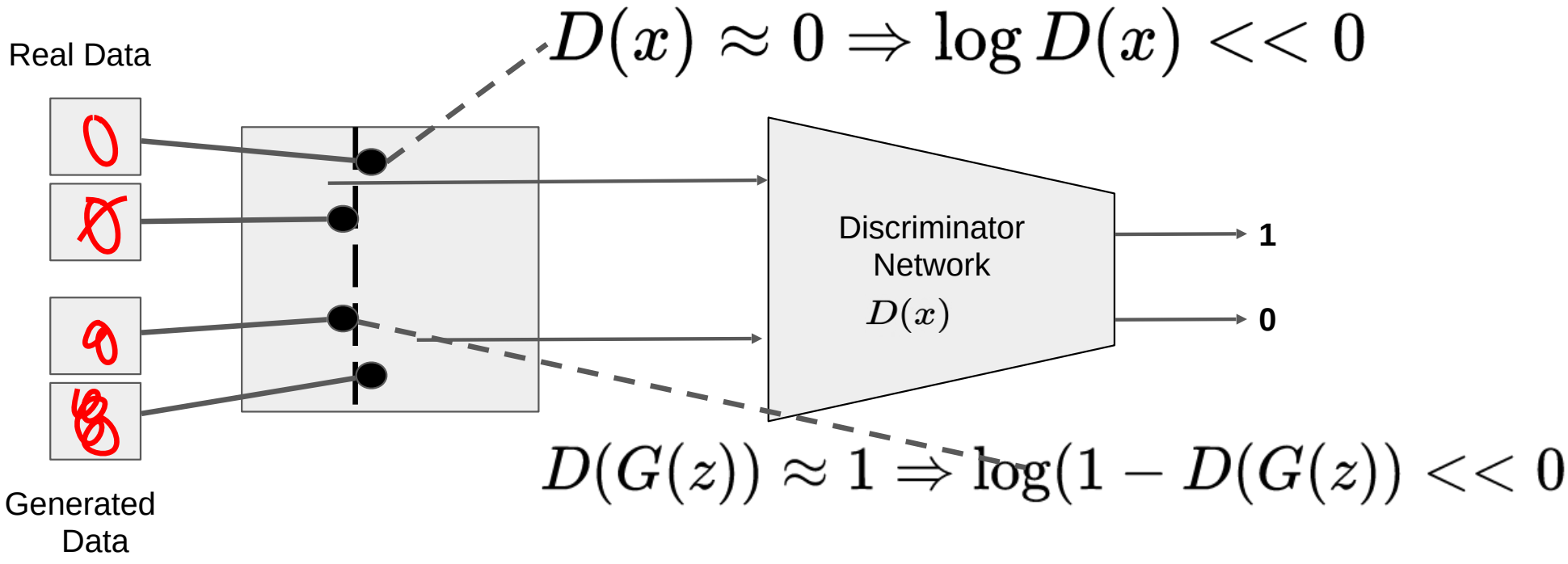
Maximizing this loss

Training Discriminator Network

Before Training

$$\max_D \left[\mathbb{E}_{x \sim P_{data}(x)} log(D(x)) + \mathbb{E}_{z \sim P_z(z)} log(1 - D(G(z))) \right]$$

$D(x)$ should be 1 $D(G(z))$ should be 0

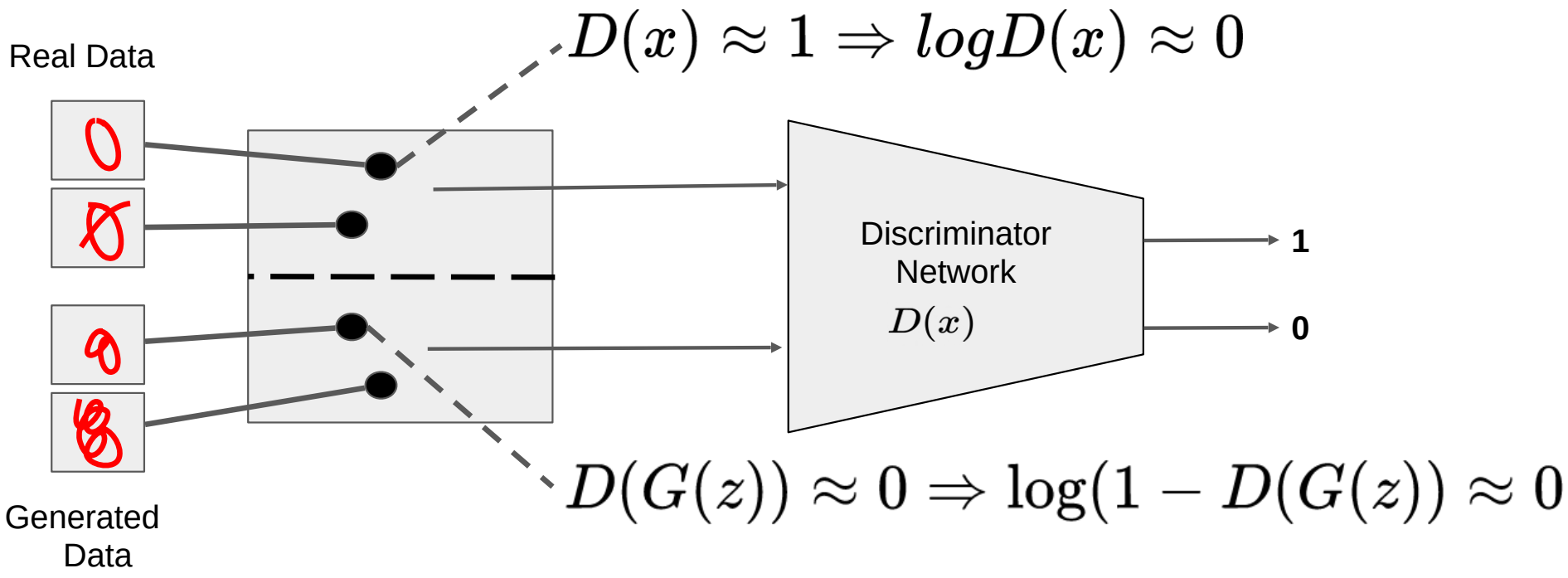


Training Discriminator Network

After Training

$$\max_D \left[\mathbb{E}_{x \sim P_{data}(x)} log(D(x)) + \mathbb{E}_{z \sim P_z(z)} log(1 - D(G(z)))\right]$$

$D(x)$ should be 1 $D(G(z))$ should be 0



Training Generator Network

Non-saturating Game - Heuristic

$$\min_G [\mathbb{E}_{x \sim P_{data}(x)} log D(x) + \mathbb{E}_{z \sim P_z(z)} log(1 - D(G(z)))]$$

There is no ***G*** in this term

$$\Rightarrow \min_G [\mathbb{E}_{z \sim P_z(z)} log(1 - D(G(z)))]$$

$D(G(z))$ should not be **0** $\Rightarrow D(G(z))$ should be **1**

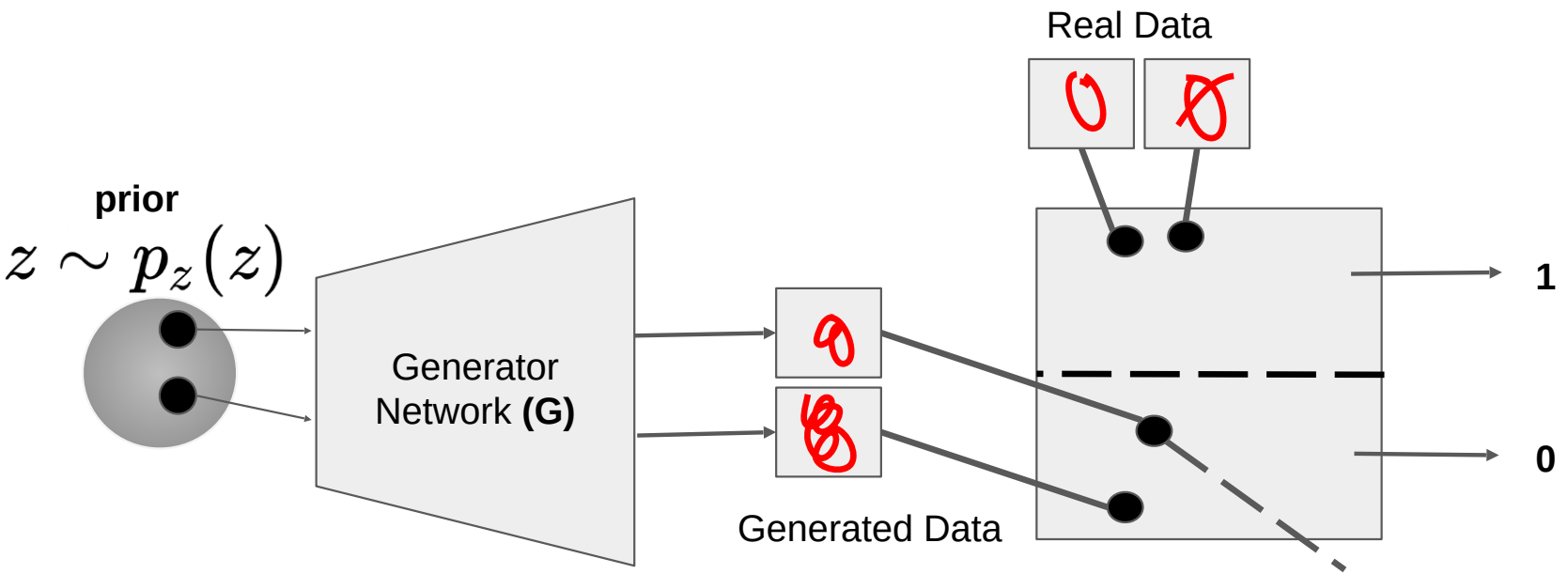
$$\Rightarrow \max_G [\mathbb{E}_{z \sim P_z(z)} log(1 - D(G(z)))]$$

Training Generator Network

Before Training

$$\max_G [\mathbb{E}_{z \sim P_z(z)} log(D(G(z)))]$$

$D(G(z))$ should be **1**



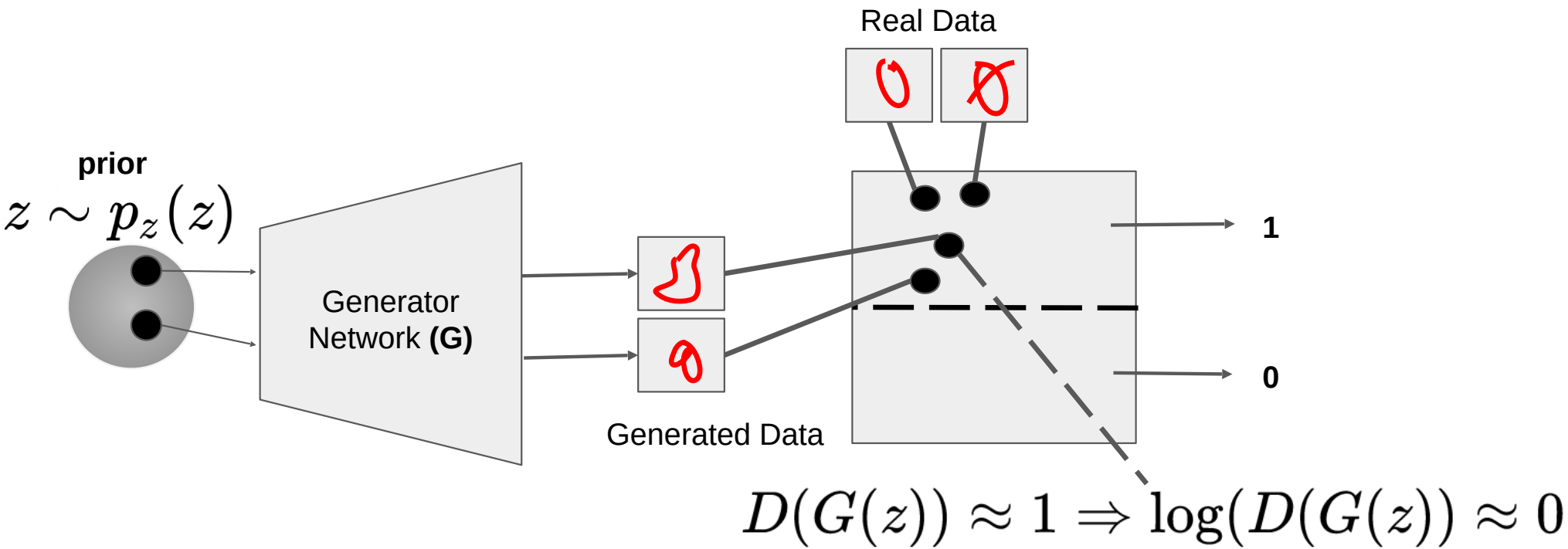
$$D(G(z)) \approx 0 \Rightarrow \log(D(G(z)) << 0$$

Training Generator Network

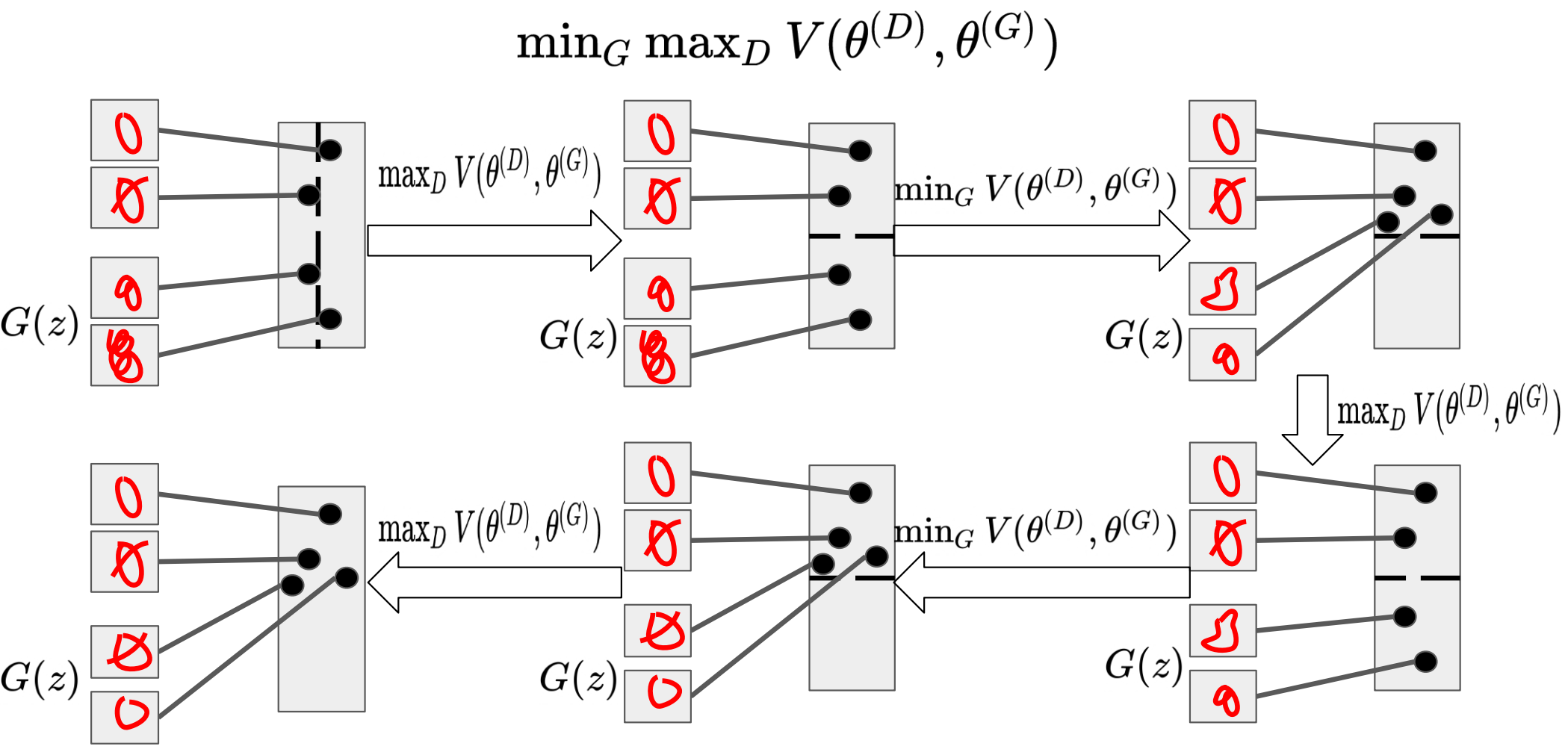
After Training

$$\max_G \left[\mathbb{E}_{z \sim P_z(z)} \boxed{log(D(G(z)))} \right]$$

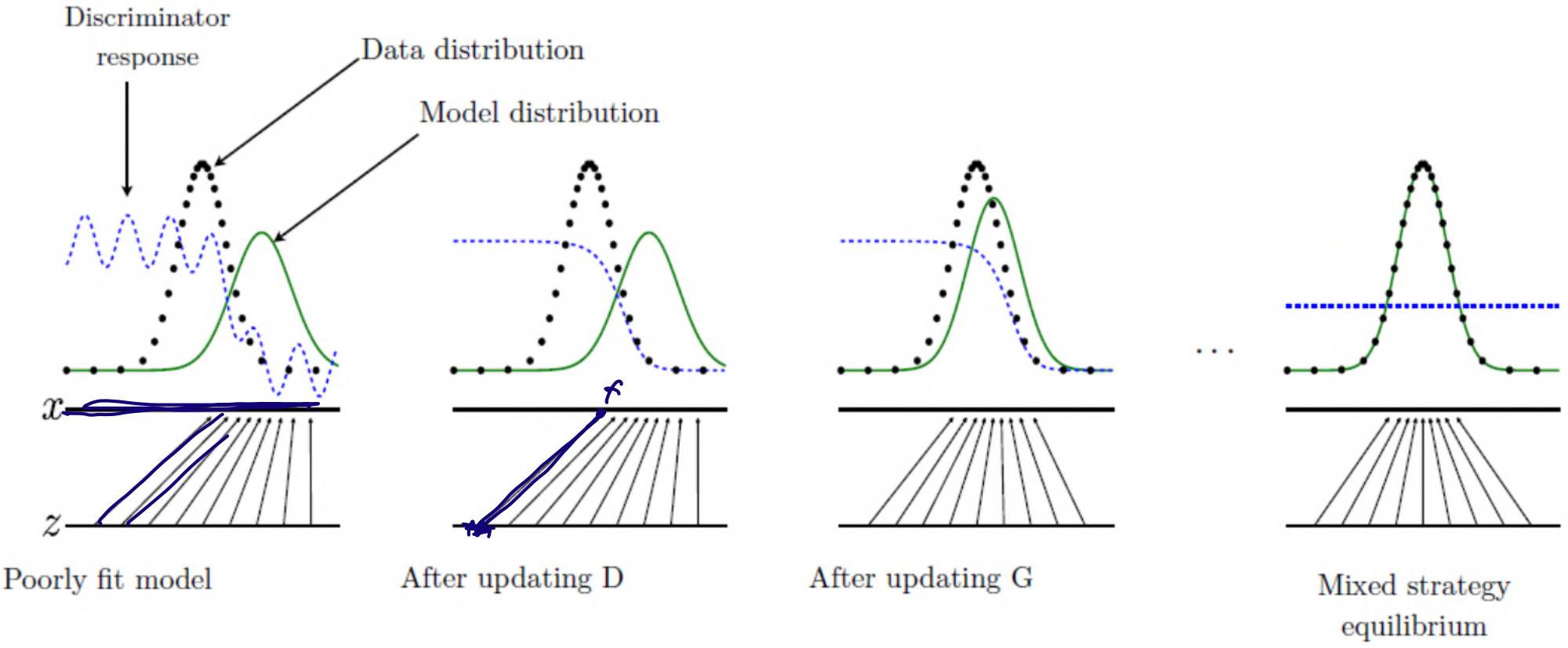
$D(G(z))$ should be **1**



Training Generative Adversarial Network



Learning Process



Goodfellow et al, Generative Adversarial Networks, Advances in Neural Information Processing Systems 27, 2014

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

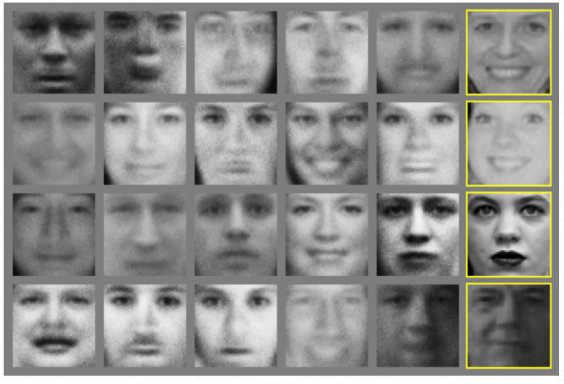
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

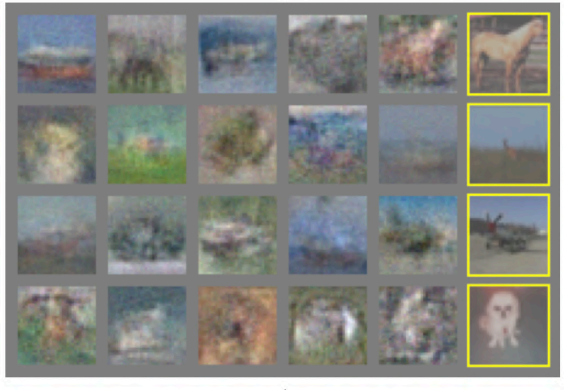
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Generated data

a)

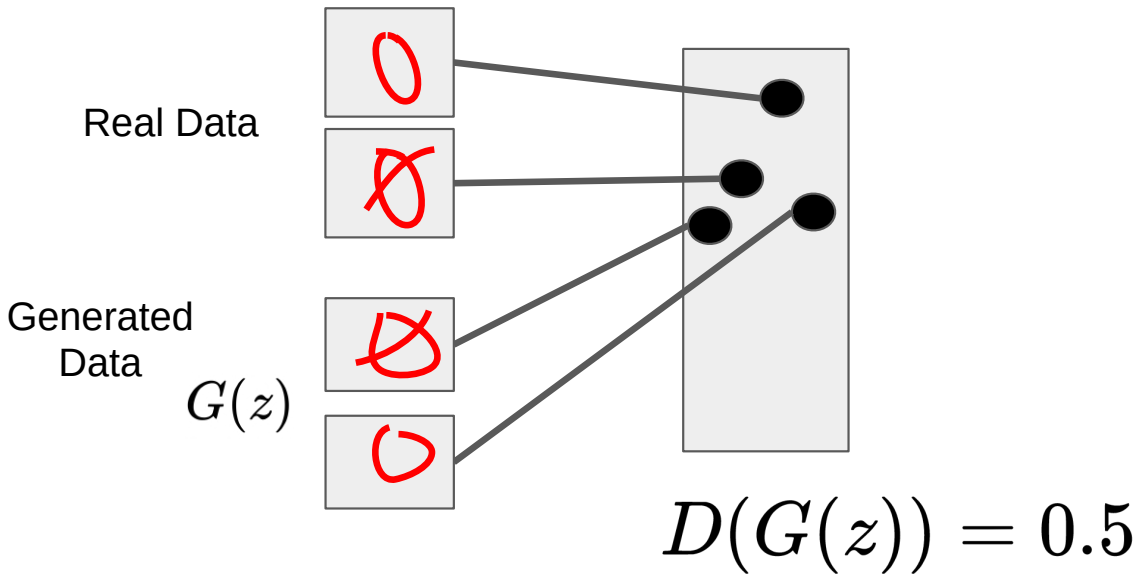


b)



Global Optimum

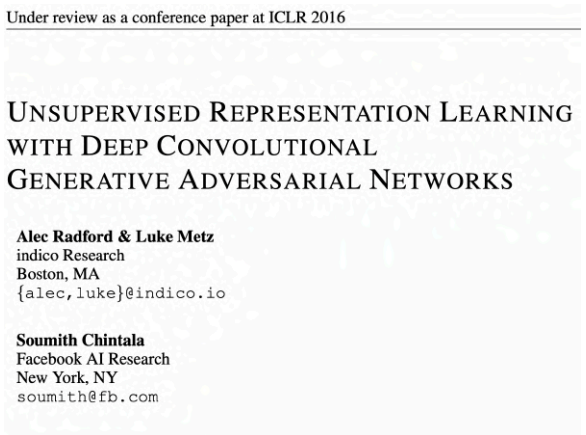
$$p_{data} = p_{generator}$$



Deep Convolutional GAN (DCGAN)

DCGAN uses CNN architecture as stable architecture to train GAN. It is achieved by adopting certain architectural constraint to GAN in following ways:

1. Replace pooling layers with strided convolutions in discriminator and transposed convolutions in generator.
2. Use Batch Normalization in both the generator and the discriminator.
3. Remove fully connected layers.
4. Use ReLU activation in generator for all layers except for the output, which uses tanh.
5. Use LeakyReLU activation in the discriminator for all layers.



Under review as a conference paper at ICLR 2016

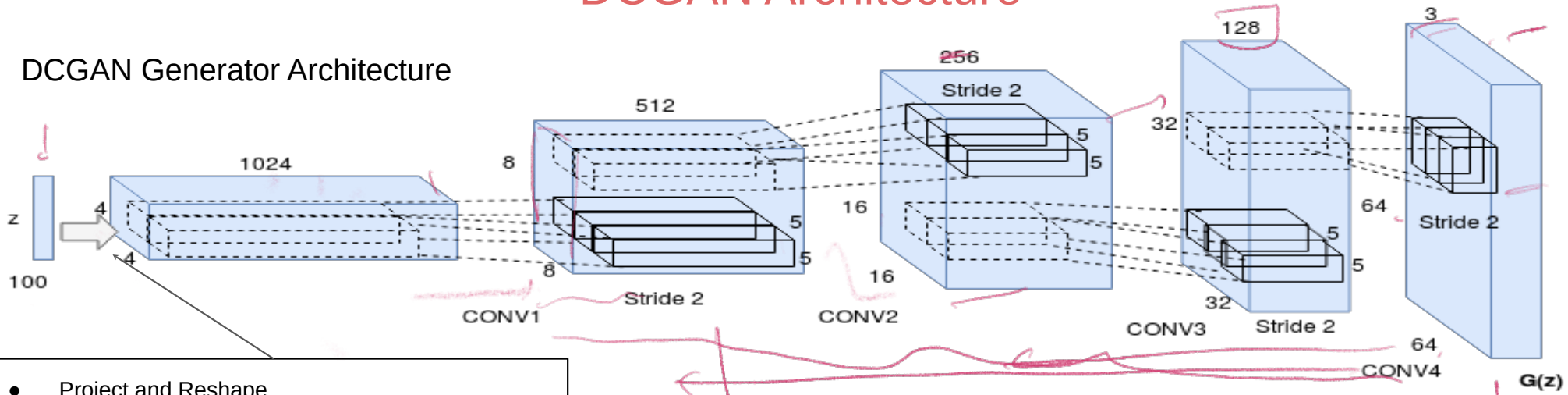
UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

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

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

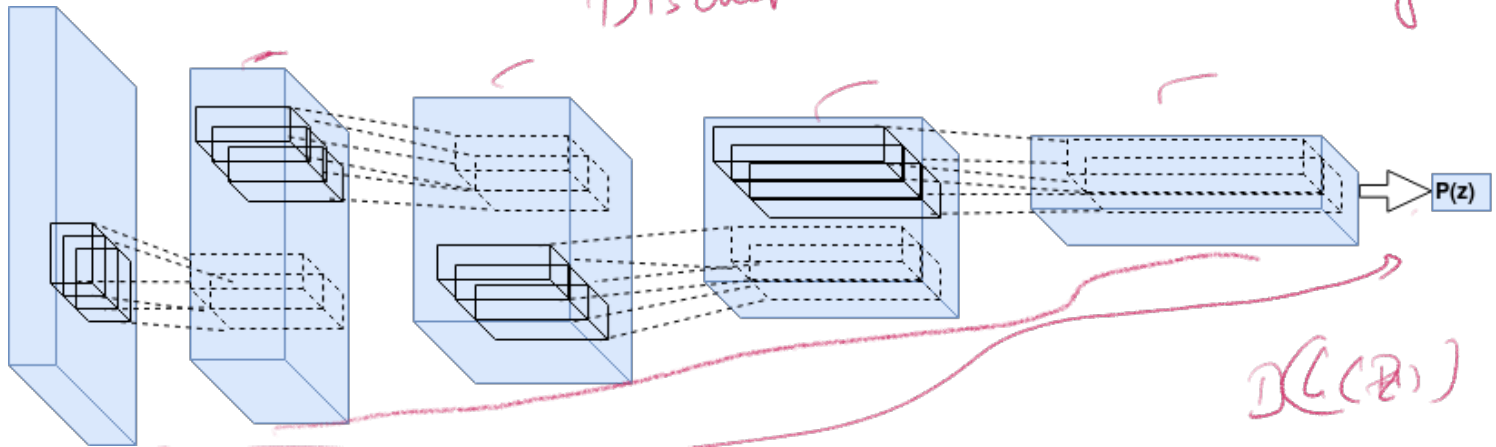
DCGAN Architecture

DCGAN Generator Architecture



- Project and Reshape
- Z vector is passed through fully connected layer of output 16384.
- This output is then reshaped to 4 x 4 x 1024.

DCGAN Discriminator Architecture



DCGAN Results

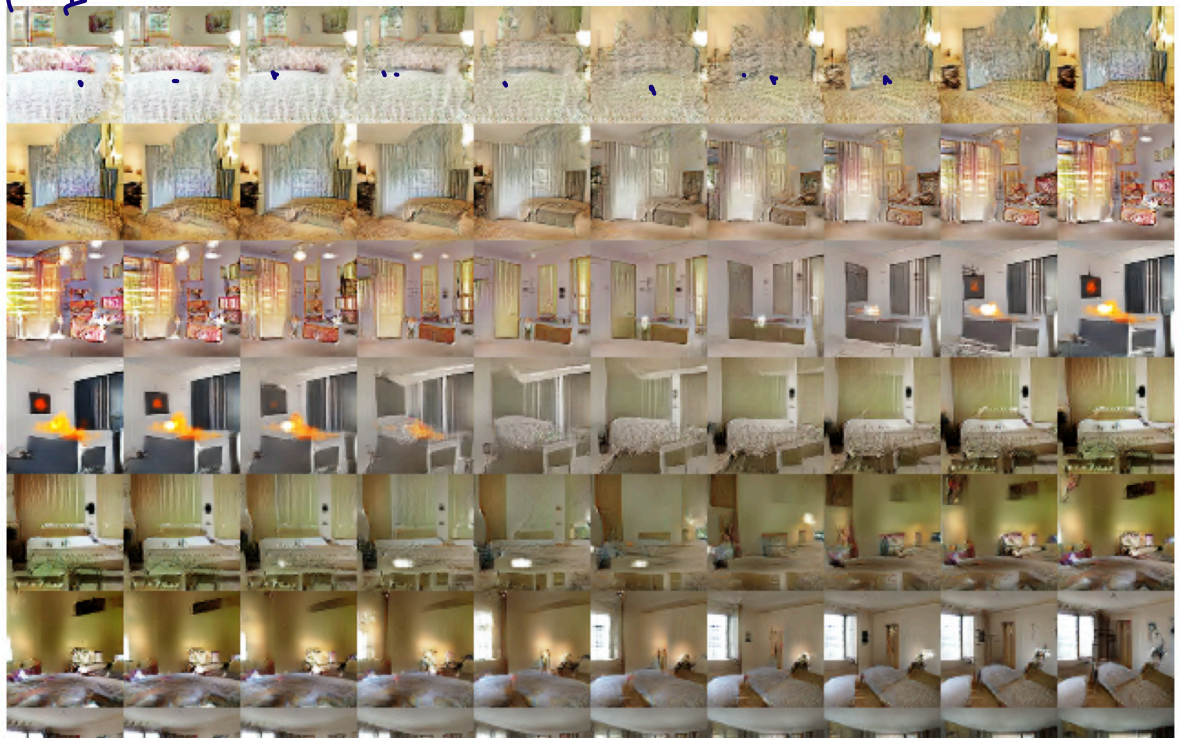


Figure 4: Top rows: Interpolation between a series of 9 random points in Z show that the space learned has smooth transitions, with every image in the space plausibly looking like a bedroom. In

DC-GAN Results

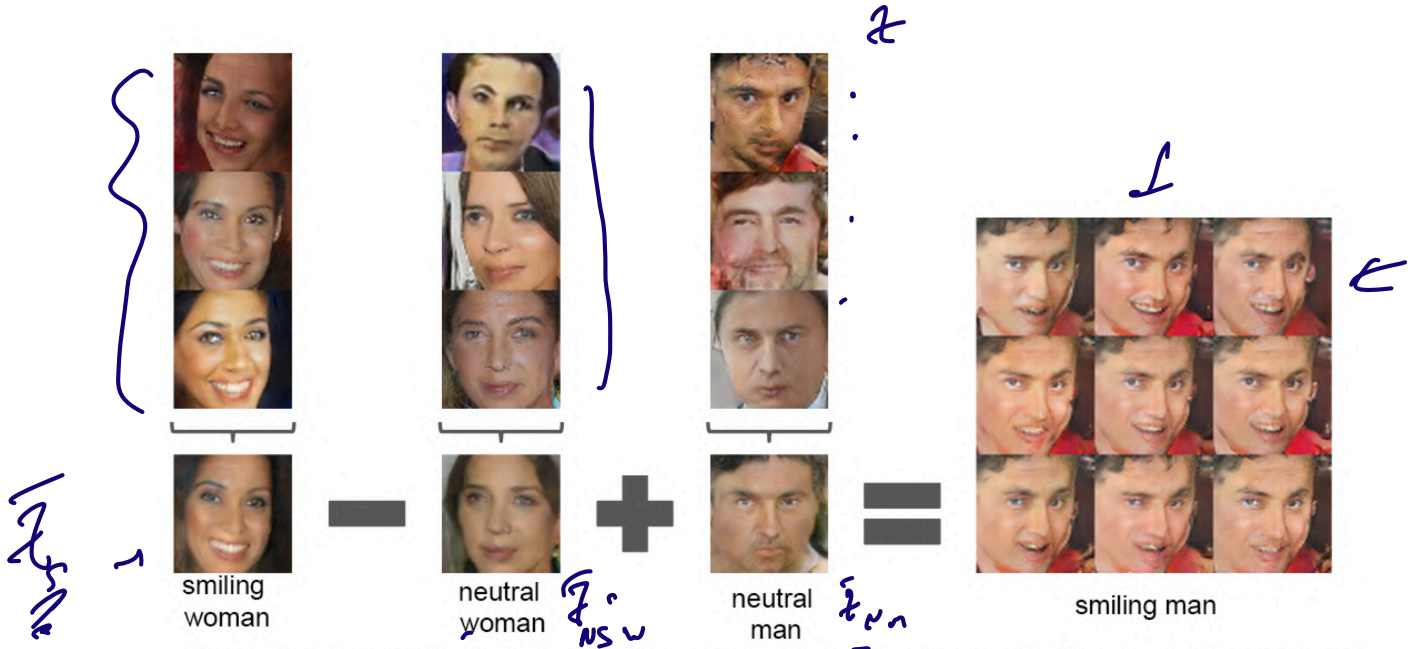


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples)

DC-GAN Results

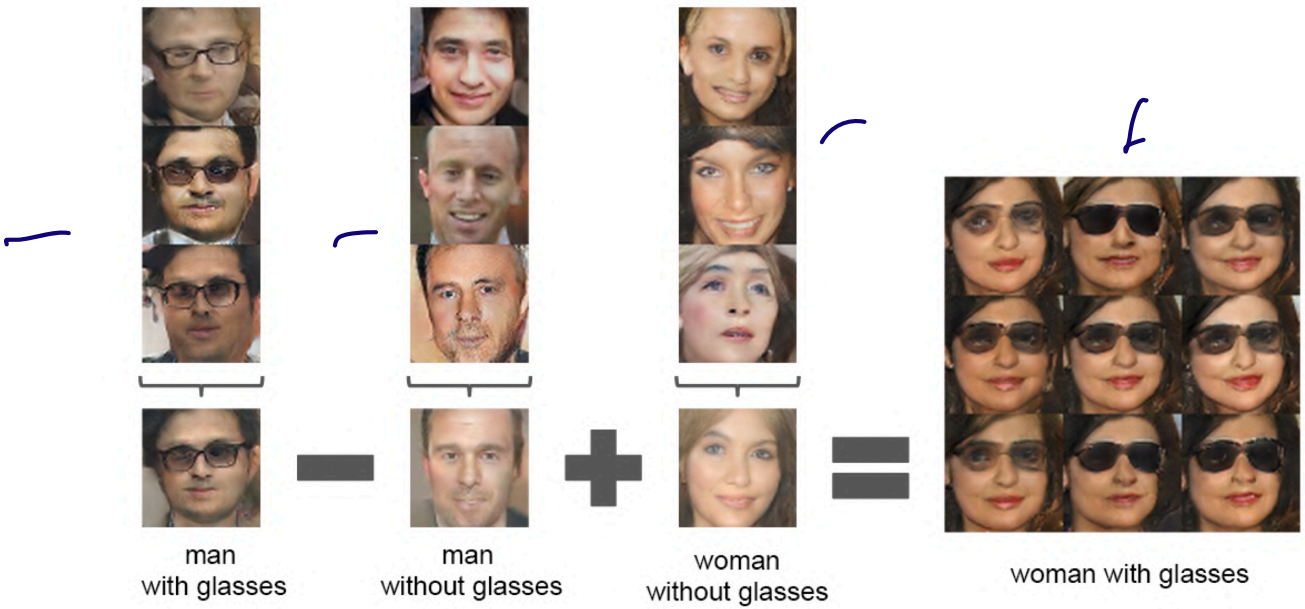
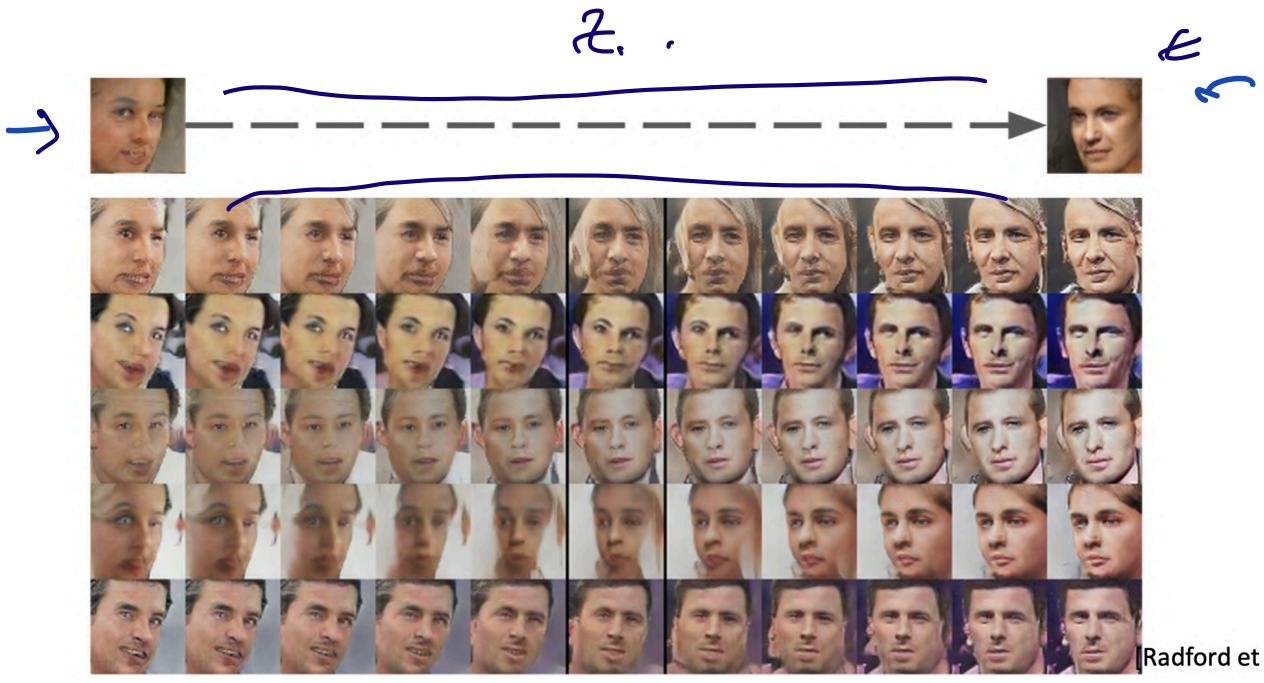


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples. Applying arithmetic in the input space (bottom two examples)

DCGAN Results

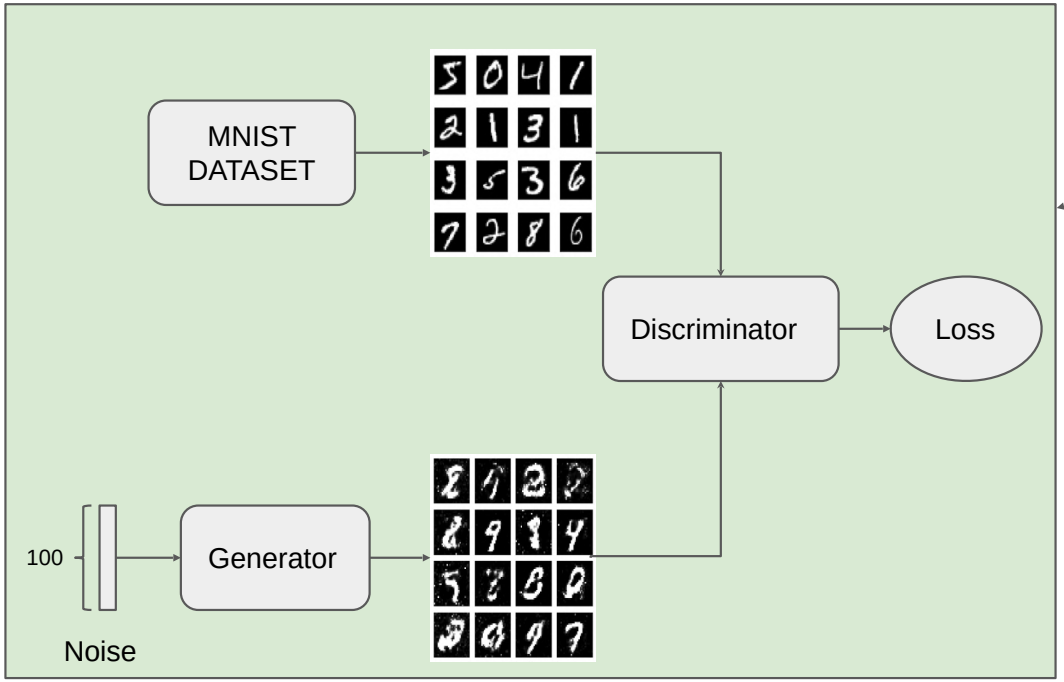


[Radford et al 2016]

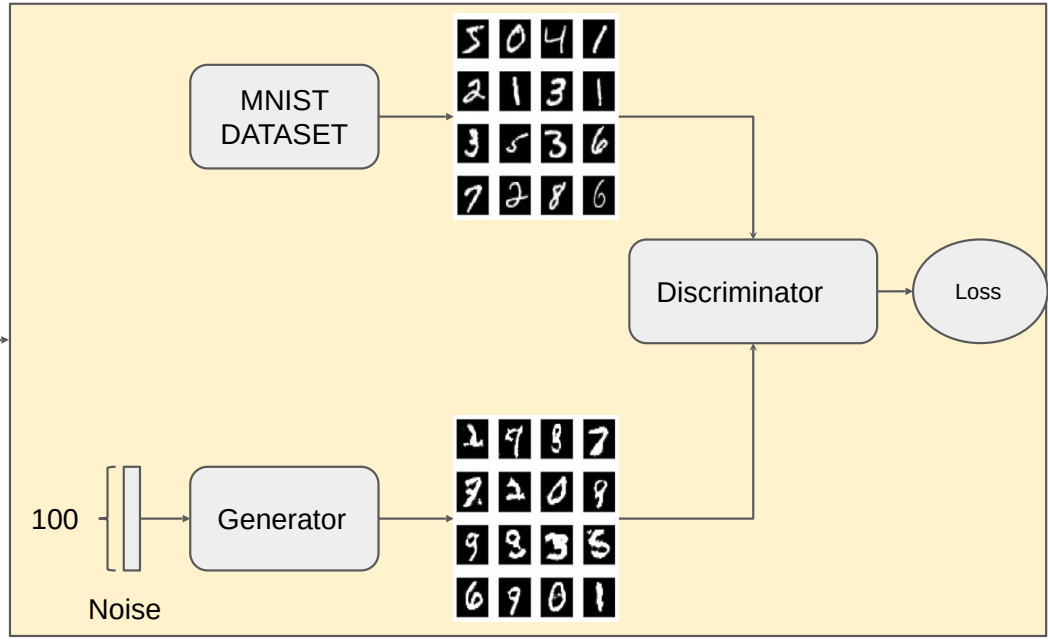
Figure 8: A ”turn” vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

DCGAN with MNIST

Data generation by GAN



Data generation by DCGAN



Progressive Growing of GANs

PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION

Tero Karras NVIDIA
Timo Aila NVIDIA
Samuli Laine NVIDIA
Jaakko Lehtinen NVIDIA and Aalto University
{tkarras,taila,slaine,jlehtinen}@nvidia.com

[cs.NE] 26 Feb 2018

ABSTRACT

We describe a new training methodology for generative adversarial networks. The key idea is to grow both the generator and discriminator progressively: starting from a low resolution, we add new layers that model increasingly fine details as training progresses. This both speeds the training up and greatly stabilizes it, allowing us to produce images of unprecedented quality, e.g., CELEBA images at 1024^2. We also propose a simple way to increase the variation in generated images, and achieve a record inception score of 8.80 in unsupervised CIFAR10. Additionally, we describe several implementation details that are important for discouraging unhealthy competition between the generator and discriminator. Finally, we suggest a new metric for evaluating GAN results, both in terms of image quality and variation. As an additional contribution, we construct a higher-quality version of the CELEBA dataset.

Progressive GANs

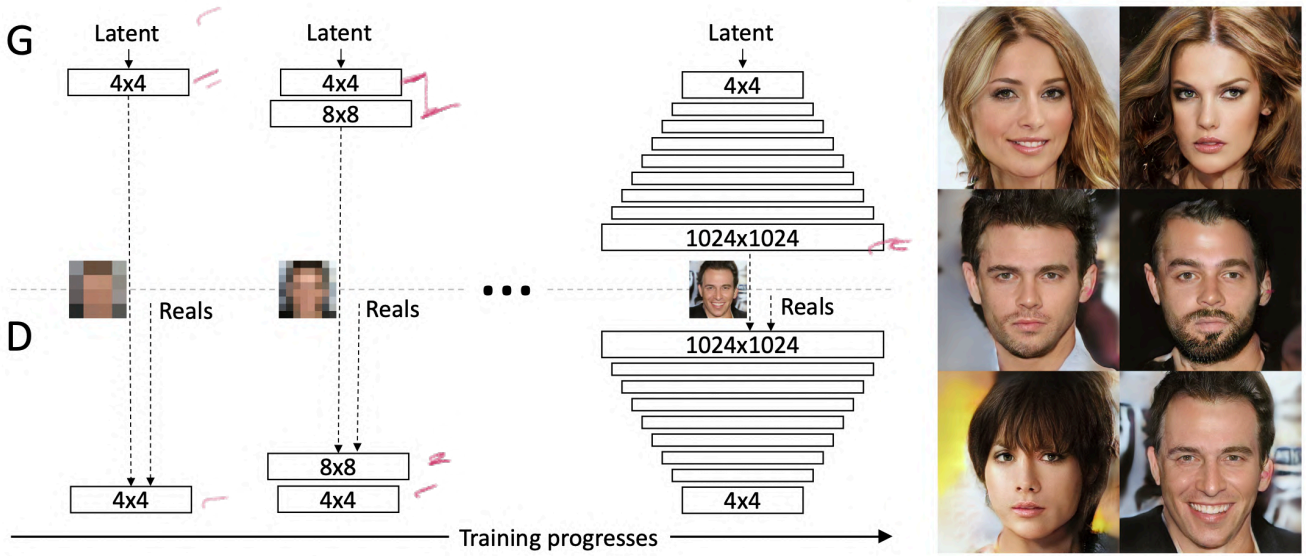


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Progressive GANs

Our (256×256)

[cs.LG] 16 Feb 2018

SNGAN

ABSTRACT

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

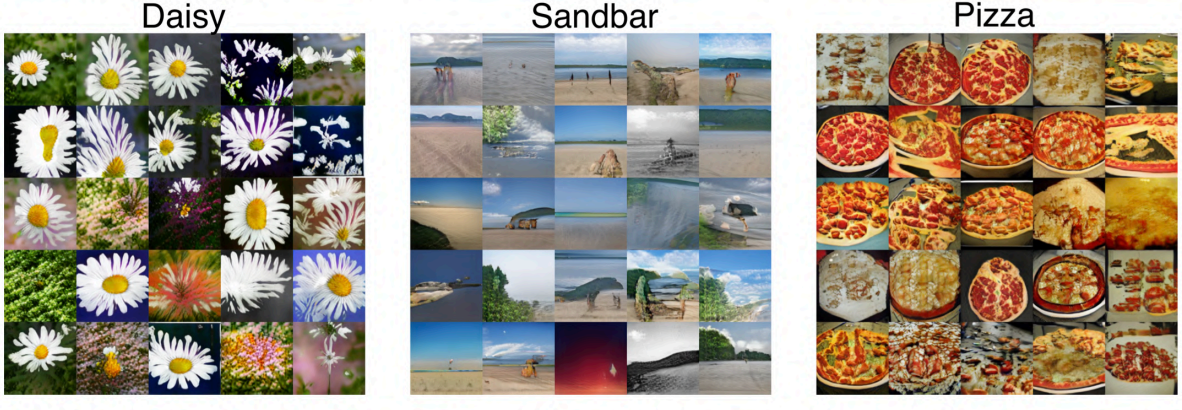
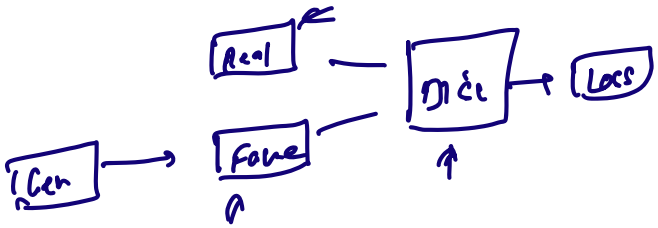


Figure 7: 128x128 pixel images generated by SN-GANs trained on ILSVRC2012 dataset. The

SNGAN



Algorithm 1 SGD with spectral normalization

- Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
- For each update and each layer l:
  1. Apply power iteration method to a unnormalized weight W^l:

$$\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l\|_2 \tag{20}$$

$$\tilde{\boldsymbol{u}}_l \leftarrow W^l \tilde{\boldsymbol{v}}_l / \|W^l \tilde{\boldsymbol{v}}_l\|_2 \tag{21}$$

  2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:

$$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l) \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}} W^l \tilde{\boldsymbol{v}}_l \tag{22}$$

  3. Update W^l with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate α:

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M) \tag{23}$$

Self Attention GAN (SAGAN)

Self-Attention Generative Adversarial Networks

Han Zhang [1 2] **Ian Goodfellow** [3] **Dimitris Metaxas** [1] **Augustus Odena** [2]

Abstract

In this paper, we propose the Self-Attention Generative Adversarial Network (SAGAN) which allows attention-driven, long-range dependency modeling for image generation tasks. Traditional convolutional GANs generate high-resolution details as a function of only spatially local points in lower-resolution feature maps. In SAGAN, details can be generated using cues from all feature locations. Moreover, the discriminator can check that highly detailed features in distant portions of the image are consistent with each other. Furthermore, recent work has shown that generator conditioning affects GAN performance. Leveraging this insight, we apply spectral normalization to the GAN generator and find that this improves training dynamics. The proposed SAGAN performs better than prior work[1], boosting the best published Inception score from 36.8 to 52.52 and reducing Fréchet Inception distance from 27.62 to 18.65 on the challenging ImageNet dataset. Visualization of the attention layers shows that the generator leverages neighborhoods that correspond to object shapes rather than local regions of fixed shape.

SAGAN

Self-Attention Generative Adversarial Networks

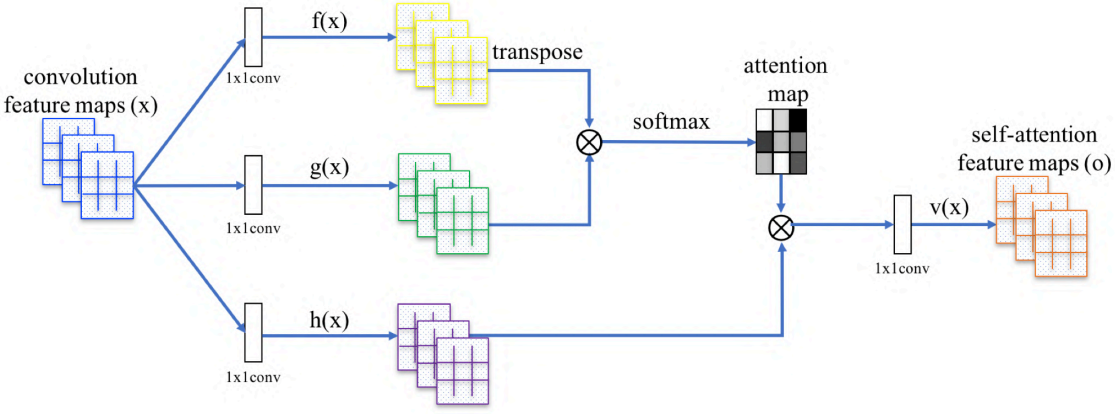


Figure 2. The proposed self-attention module for the SAGAN. The ⊗ denotes matrix multiplication. The softmax operation is performed on each row.

SAGAN

The image features from the previous hidden layer $\boldsymbol{x} \in \mathbb{R}^{C \times N}$ are first transformed into two feature spaces $\boldsymbol{f}, \boldsymbol{g}$ to calculate the attention, where $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W_f x},\ \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{W_g x}$

$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})},\ \text{where } s_{ij} = \boldsymbol{f}(\boldsymbol{x_i})^T \boldsymbol{g}(\boldsymbol{x_j}),\quad (1)$$

and $\beta_{j,i}$ indicates the extent to which the model attends to the i^{th} location when synthesizing the j^{th} region. Here, C is the number of channels and N is the number of feature locations of features from the previous hidden layer. The output of the attention layer is $\boldsymbol{o} = (\boldsymbol{o_1}, \boldsymbol{o_2}, ..., \boldsymbol{o_j}, ..., \boldsymbol{o_N}) \in \mathbb{R}^{C \times N}$, where,

$$\boldsymbol{o_j} = \boldsymbol{v}\left(\sum_{i=1}^{N} \beta_{j,i} \boldsymbol{h}(\boldsymbol{x_i})\right),\ \boldsymbol{h}(\boldsymbol{x_i}) = \boldsymbol{W_h x_i},\ \boldsymbol{v}(\boldsymbol{x_i}) = \boldsymbol{W_v x_i} \quad (2)$$

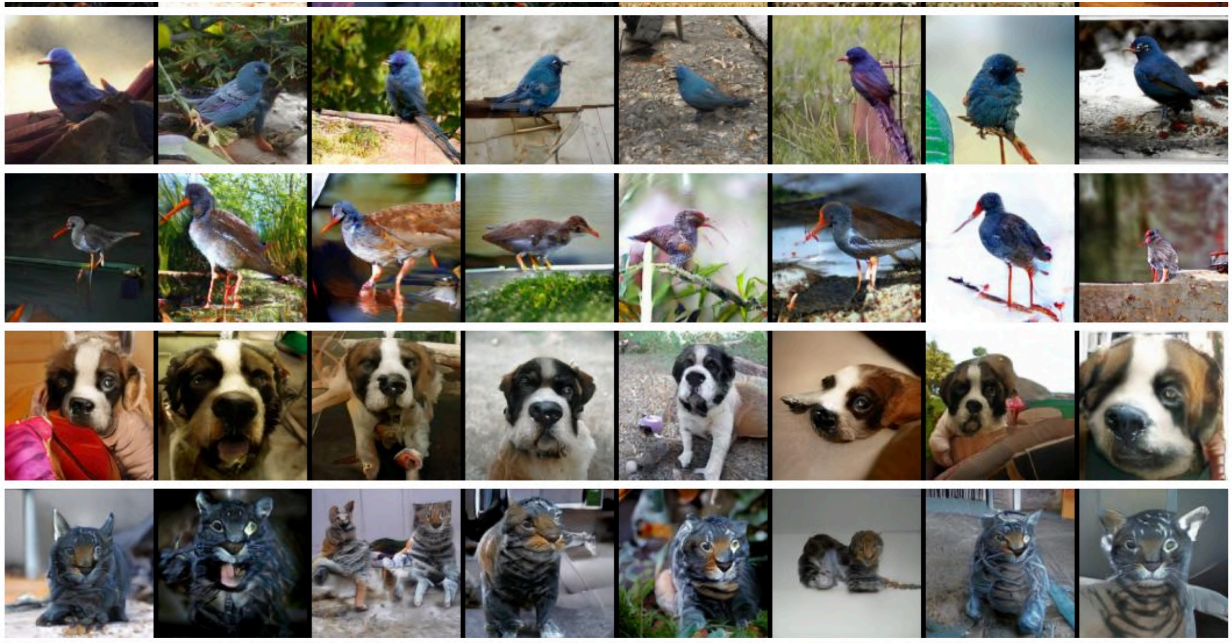
In the above formulation, $\boldsymbol{W_g} \in \mathbb{R}^{\bar{C} \times C}$, $\boldsymbol{W_f} \in \mathbb{R}^{\bar{C} \times C}$, $\boldsymbol{W_h} \in \mathbb{R}^{\bar{C} \times C}$, and $\boldsymbol{W_v} \in \mathbb{R}^{C \times \bar{C}}$ are the learned weight matrices, which are implemented as 1×1 convolutions. Since We did not notice any significant performance decrease when reducing the channel number of $\bar{C}$ to be C/k, where $k = 1, 2, 4, 8$ after few training epochs on ImageNet. For memory efficiency, we choose $k = 8$ (*i.e.*, $\bar{C} = C/8$) in all our experiments.

In addition, we further multiply the output of the attention layer by a scale parameter and add back the input feature map. Therefore, the final output is given by,

$$\boldsymbol{y_i} = \gamma \boldsymbol{o_i} + \boldsymbol{x_i}, \quad (3)$$

where γ is a learnable scalar and it is initialized as 0. In-

SAGAN



BigGAN

LARGE SCALE GAN TRAINING FOR HIGH FIDELITY NATURAL IMAGE SYNTHESIS

Andrew Brock[*†]
Heriot-Watt University
ajb5@hw.ac.uk

Jeff Donahue[†]
DeepMind
jeffdonahue@google.com

Karen Simonyan[†]
DeepMind
simonyan@google.com

LG] 25 Feb 2019

ABSTRACT

Despite recent progress in generative image modeling, successfully generating high-resolution, diverse samples from complex datasets such as ImageNet remains an elusive goal. To this end, we train Generative Adversarial Networks at the largest scale yet attempted, and study the instabilities specific to such scale. We find that applying orthogonal regularization to the generator renders it amenable to a simple "truncation trick," allowing fine control over the trade-off between sample fidelity and variety by reducing the variance of the Generator's input. Our modifications lead to models which set the new state of the art in class-conditional image synthesis. When trained on ImageNet at 128×128 resolution, our models (BigGANs) achieve an Inception Score (IS) of 166.5 and Fréchet Inception Distance (FID) of 7.4, improving over the previous best IS of 52.52 and FID of 18.65.

BigGAN

1 Introduction

Figure 1: Class-conditional samples generated by our model.

(a) 128×128 (b) 256×256 (c) 512×512

BigGAN

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

where W is a weight matrix and β a hyperparameter. This regularization is known to often be too limiting (Miyato et al., 2018), so we explore several variants designed to relax the constraint while still imparting the desired smoothness to our models. The version we find to work best removes the diagonal terms from the regularization, and aims to minimize the pairwise cosine similarity between filters but does not constrain their norm:

$$R_\beta(W) = \beta \|W^\top W \odot (\mathbf{1} - I)\|_{\mathrm{F}}^2, \tag{3}$$

where $\mathbf{1}$ denotes a matrix with all elements set to 1. We sweep β values and select 10^{-4}, finding

BigGAN

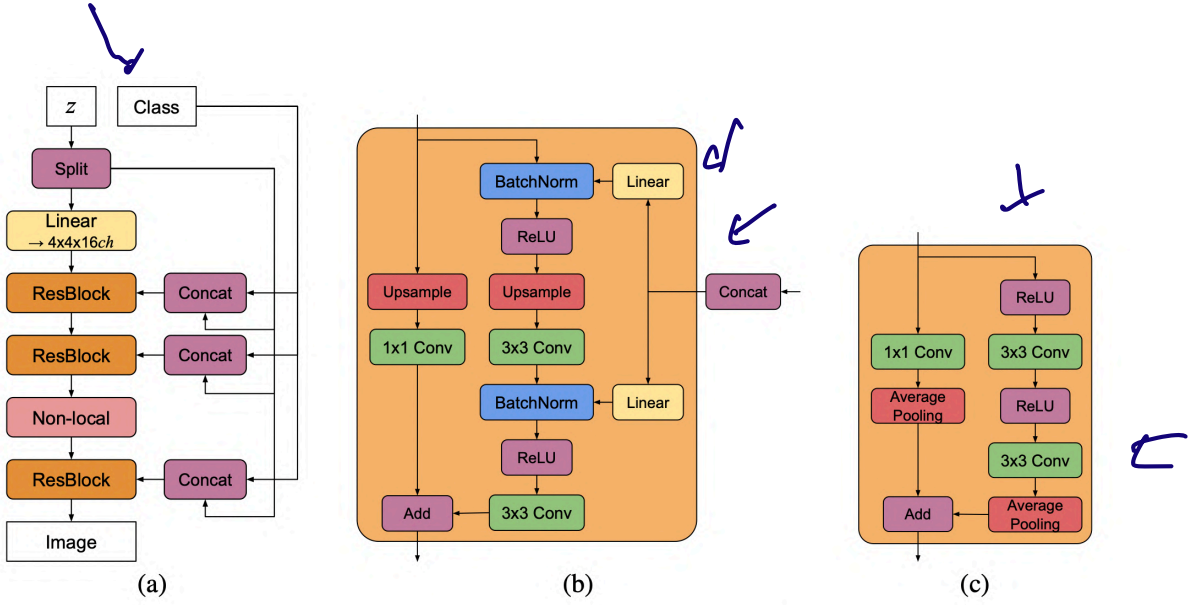
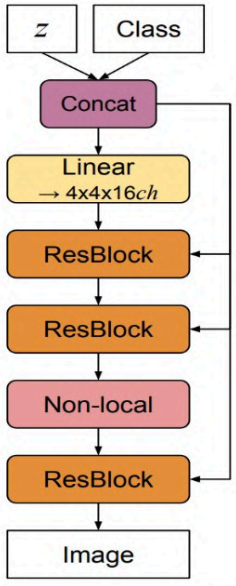
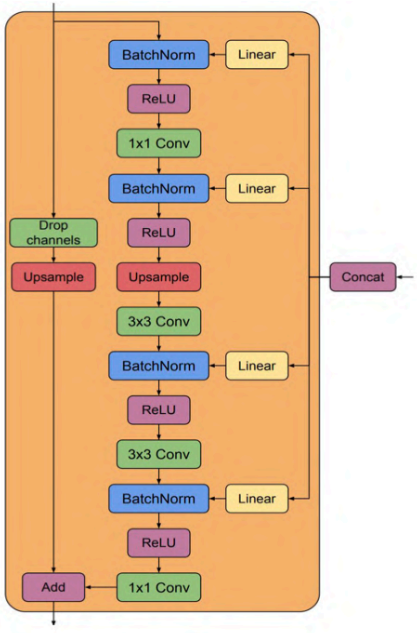
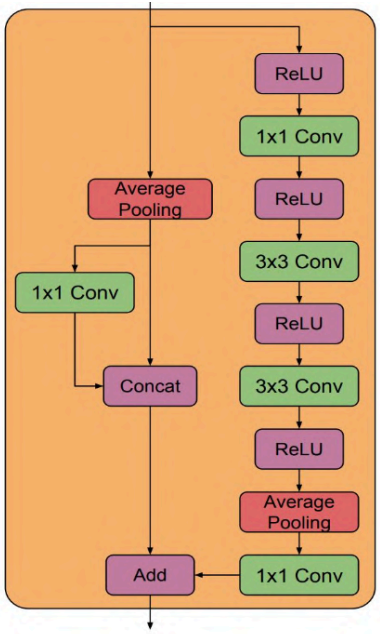


Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

BigGAN Deep


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


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


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

BigGAN Deep

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

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

(a) Generator

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

(b) Discriminator

BigGAN

- Increase your batch size (as much as you can)
- Use Cross-Replica (Sync) Batch Norm
- Increase your model size
- Wider helps as much as deeper
- Fuse class information at all levels
- Hinge Loss
- Orthonormal regularization & Truncation Trick

StyleGAN

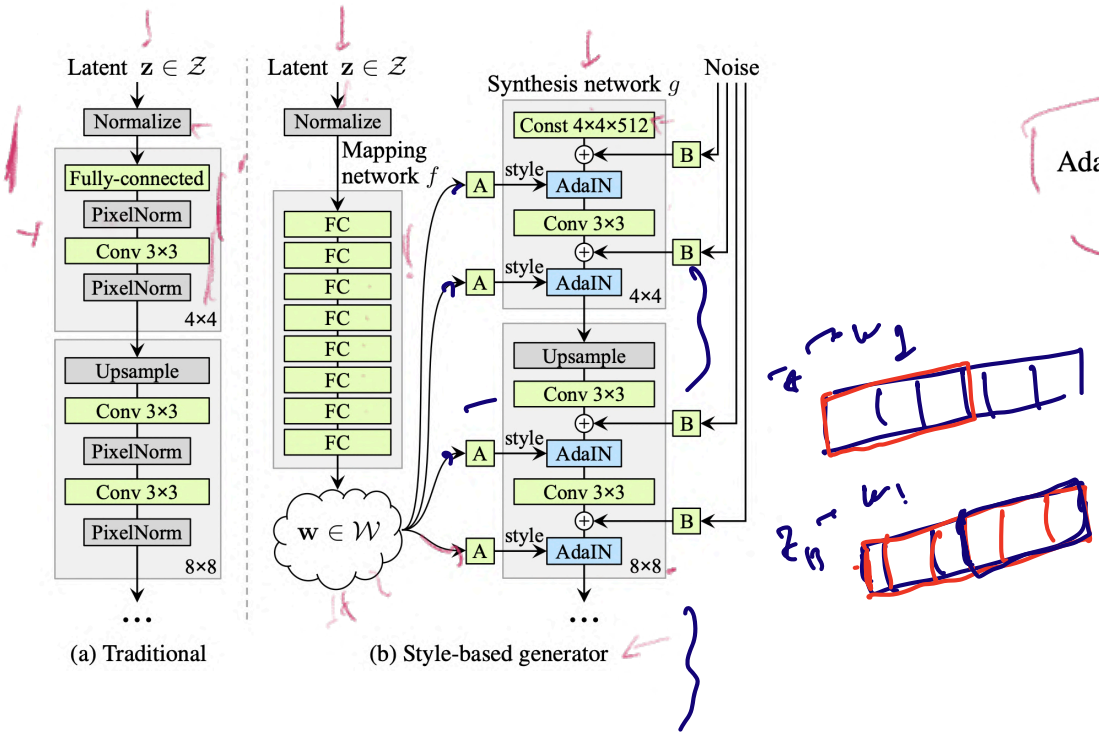
neural style transfer

4948v3 [cs.NE] 29 Mar 2019

Abstract

We propose an alternative generator architecture for generative adversarial networks, borrowing from style transfer literature. The new architecture leads to an automatically learned, unsupervised separation of high-level attributes (e.g., pose and identity when trained on human faces) and stochastic variation in the generated images (e.g., freckles, hair), and it enables intuitive, scale-specific control of the synthesis. The new generator improves the state-of-the-art in terms of traditional distribution quality metrics, leads to demonstrably better interpolation properties, and also better disentangles the latent factors of variation. To quantify interpolation quality and disentanglement, we propose two new, automated methods that are applicable to any generator architecture. Finally, we introduce a new, highly varied and high-quality dataset of human faces.

StyleGAN



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

StyleGAN Style Transfer

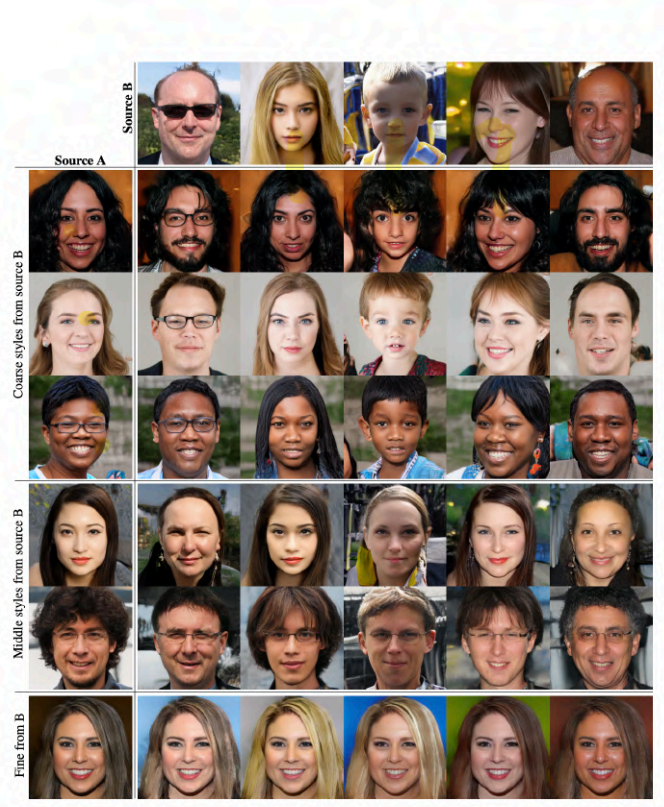


Figure 3. Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions ($4^2 - 8^2$) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions ($16^2 - 32^2$) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles ($64^2 - 1024^2$) from B brings mainly the color scheme and microstructure.

StyleGAN noise addition

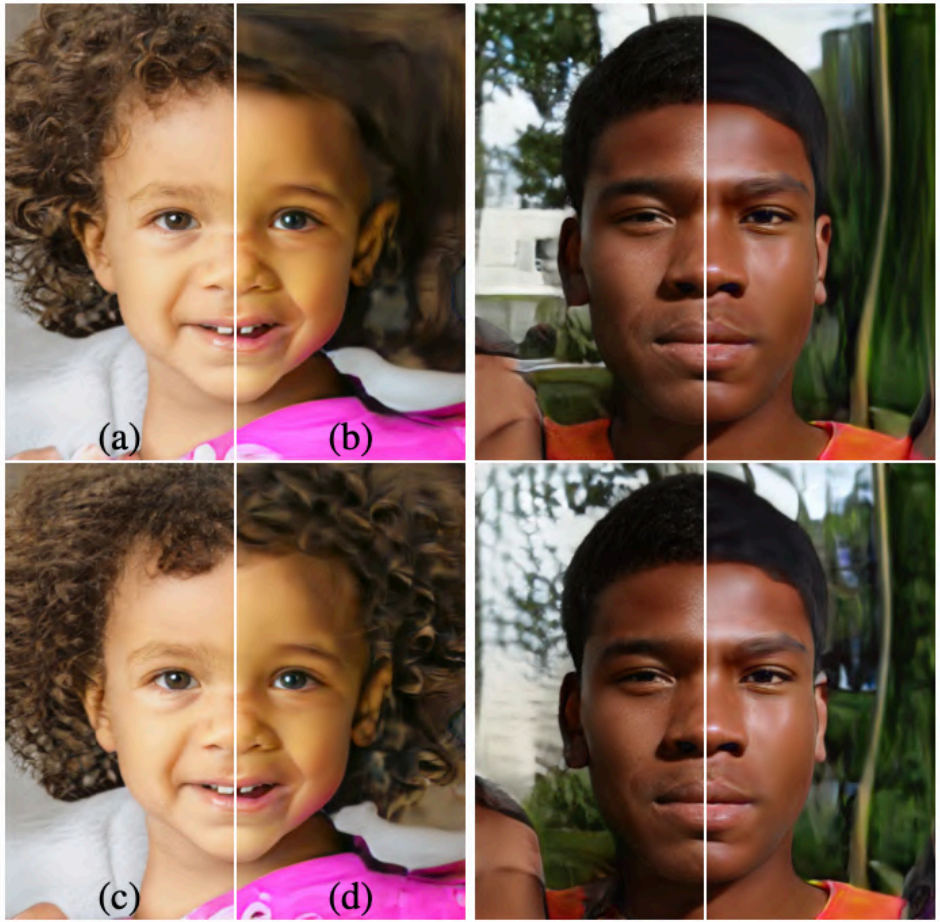


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

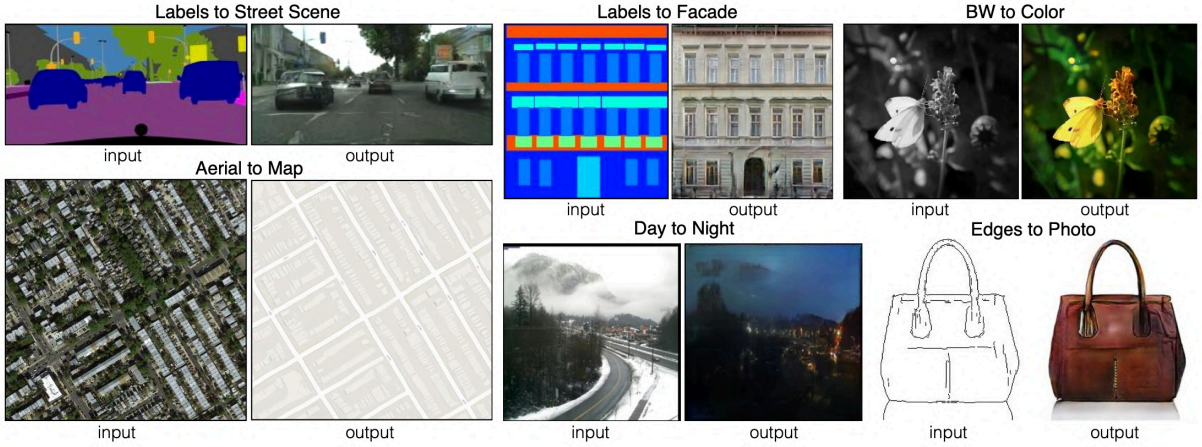
Paired Image Translation

Image-to-Image Translation with Conditional Adversarial Networks

Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros

Berkeley AI Research (BAIR) Laboratory, UC Berkeley

{isola,junyanz,tinghuiz,efros}@eecs.berkeley.edu



Paired Image Translation

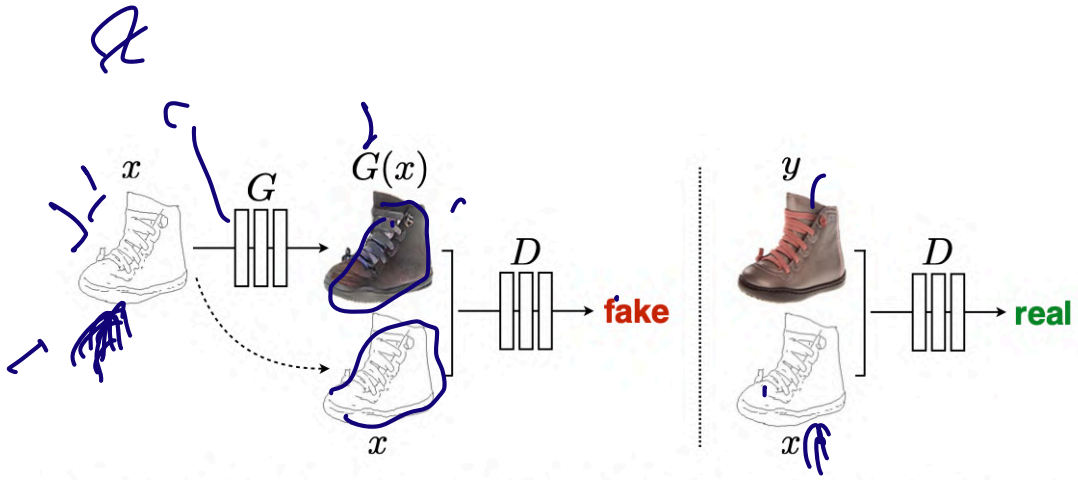


Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

Paired Image Translation

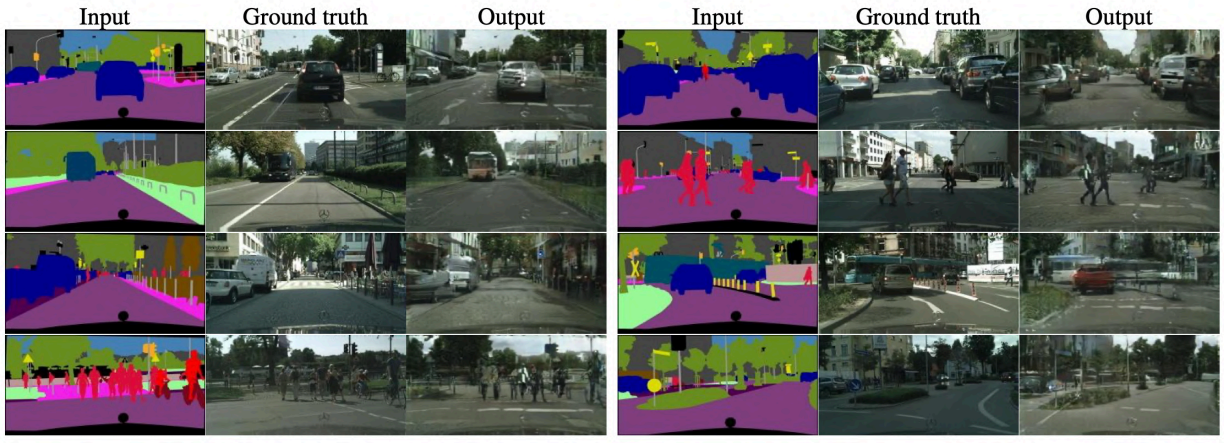


Figure 13: Example results of our method on Cityscapes labels→photo, compared to ground truth.

cycleGAN

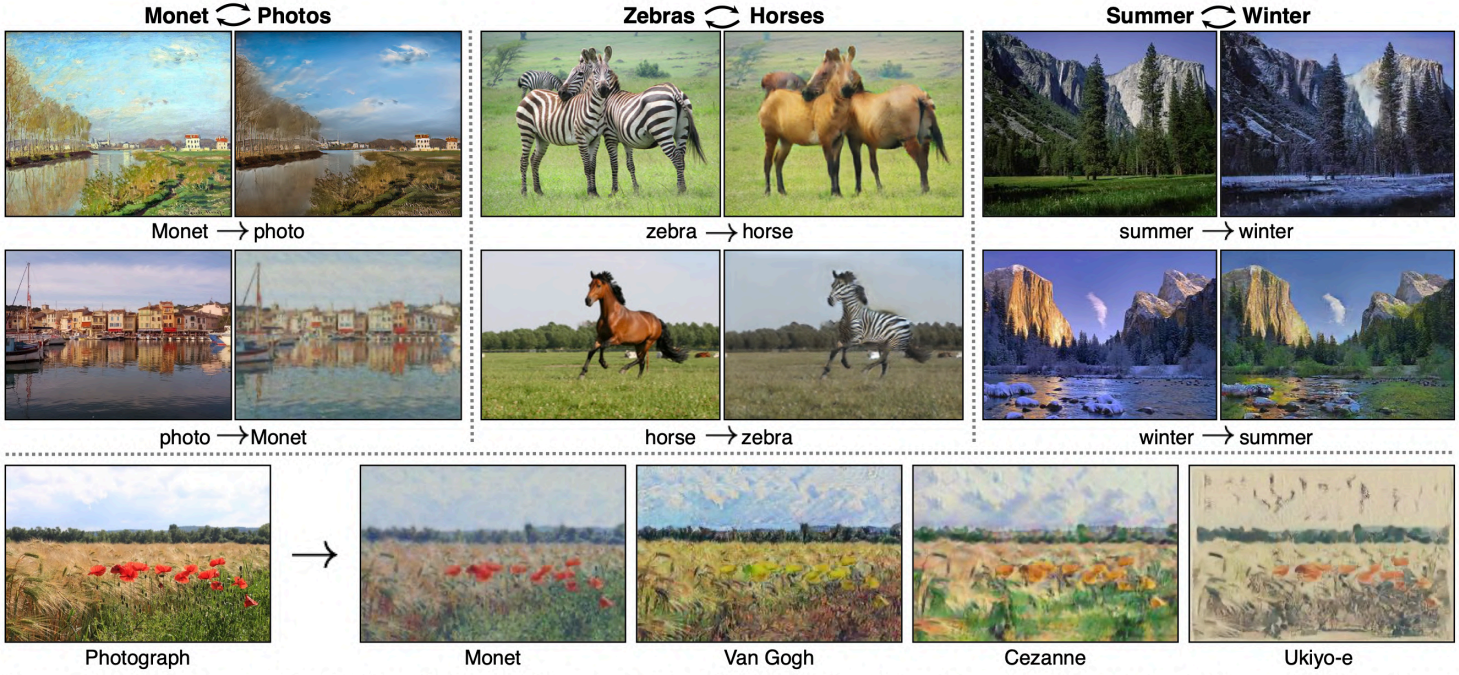


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

cycleGAN

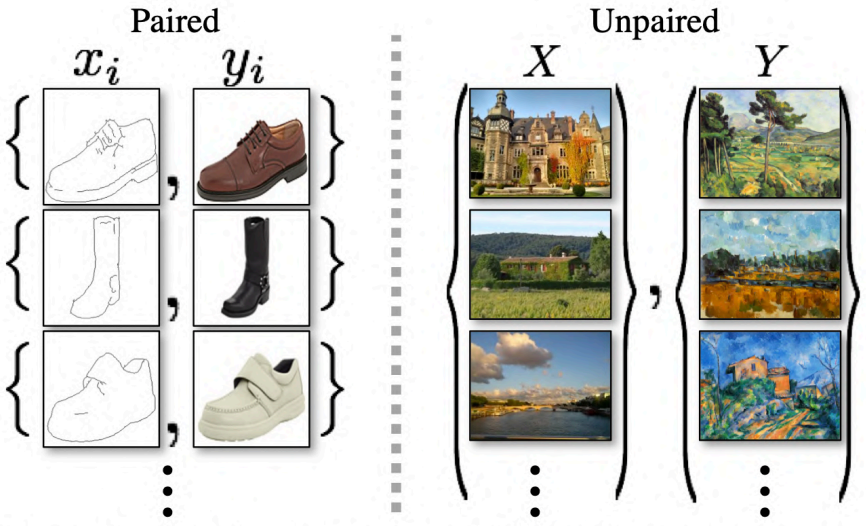


Figure 2: *Paired* training data (left) consists of training examples $\{x_i, y_i\}_{i=1}^N$, where the correspondence between x_i and y_i exists [22]. We instead consider *unpaired* training data (right), consisting of a source set $\{x_i\}_{i=1}^N$ ($x_i \in X$) and a target set $\{y_j\}_{j=1}^M$ ($y_j \in Y$), with no information provided as to which x_i matches which y_j.

cycleGAN

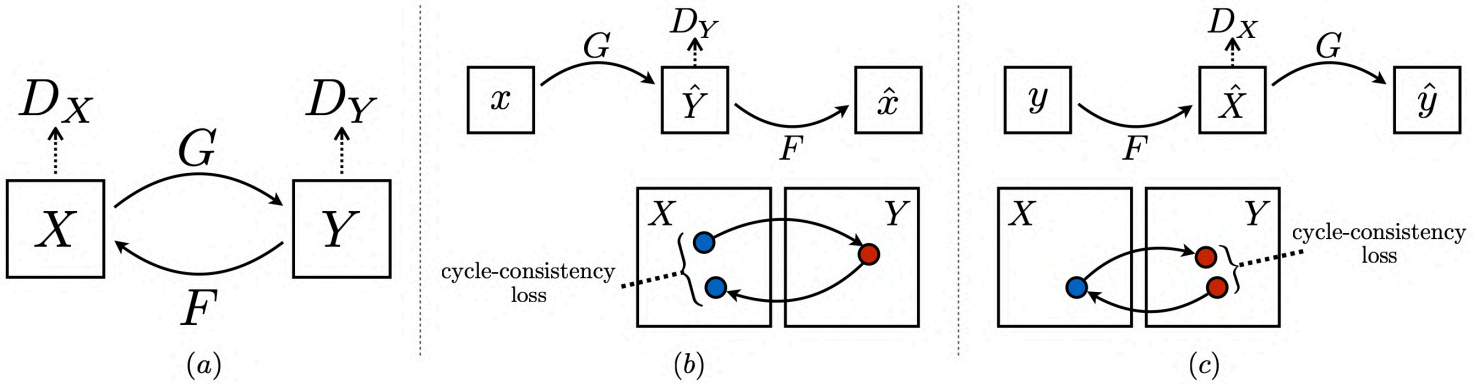


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

cycleGAN

3.1. Adversarial Loss

We apply adversarial losses [16] to both mapping functions. For the mapping function $G : X \rightarrow Y$ and its discriminator D_Y, we express the objective as:

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))], \quad (1)$$

where G tries to generate images $G(x)$ that look similar to images from domain Y, while D_Y aims to distinguish between translated samples $G(x)$ and real samples y. G aims to minimize this objective against an adversary D that tries to maximize it, i.e., $\min_G \max_{D_Y} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$. We introduce a similar adversarial loss for the mapping function $F : Y \rightarrow X$ and its discriminator D_X as well: i.e., $\min_F \max_{D_X} \mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$.

(b), for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. We call this *forward cycle consistency*. Similarly, as illustrated in Figure 3 (c), for each image y from domain Y, G and F should also satisfy *backward cycle consistency*: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. We incentivize this behavior using a *cycle consistency loss*:

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]. \quad (2)$$

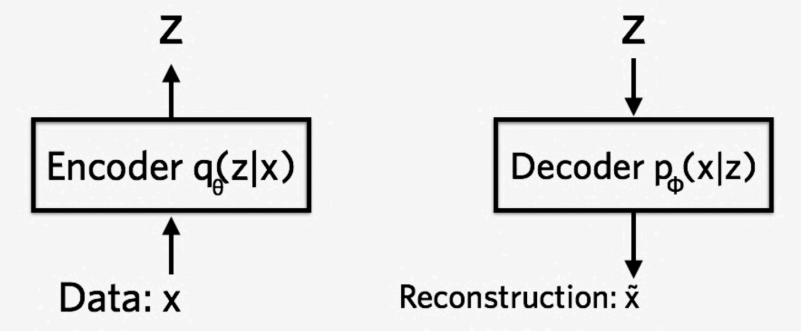
In preliminary experiments, we also tried replacing the L1 norm in this loss with an adversarial loss between $F(G(x))$ and x, and between $G(F(y))$ and y, but did not observe improved performance.

Our full objective is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F), \quad (3)$$

Variational Autoencoder

- Encoder
- Decoder
- Loss function



VAE-Encoder

- The encoder is a neural network
- Input is a datapoint x
- Output is a hidden representation z
- Parameters to be learnt are its weight and biases
- Encoder is represented as $q_\theta(z|x)$
- The output z are parameters of a Gaussian probability density

VAE Decoder

- Decoder is another neural network
- Input is the representation z
- Outputs parameters of the probability distribution of the data
- Decoder is represented by $p_\phi(x \mid z)$
- If output is a b&w image, then each pixel is either 0 or1
- Each output is then a parameter of the Bernoulli distribution

VAE Loss function

- The original information is not perfectly captured because z is not the same dimension as x
- The loss of fidelity is captured by the log-likelihood which is $log(p_\phi(x \mid z))$
- A measure of how good the reconstruction is

VAE regularizer

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

The first term is the reconstruction loss
The expectation is taken with respect to the encoder's distribution

The second term is the regulariser. This called the Kullback-Leibler divergence between the encoder's distribution and p(z)
Quantifies how close the encoder distribution is to the desired distribution p
In VAE p is represented by a standard Normal distribution with mean zero and variance of 1 p(z) = N(0,1)

KL Divergence

$$D_{KL}(p(x)||q(x)) = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)}$$

$$D_{KL}(p(x)||q(x)) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} dx$$

VAE Results

Variational Autoencoder implementation with Tensorflow.

https://github.com/wojciechmo/vae