CS230: Lecture 3
Attacking Networks with Adversarial Examples
-
Generative Adversarial Networks

Kian Katanforoosh

Today's outline

I. Attacking NNs with Adversarial Examples

II. Generative Adversarial Networks

I. Adversarial examples

Szegedy et al. (2013): several machine learning models, including state-of-the-art neural networks, are vulnerable to adversarial examples.

A. Attacking a network with adversarial examples

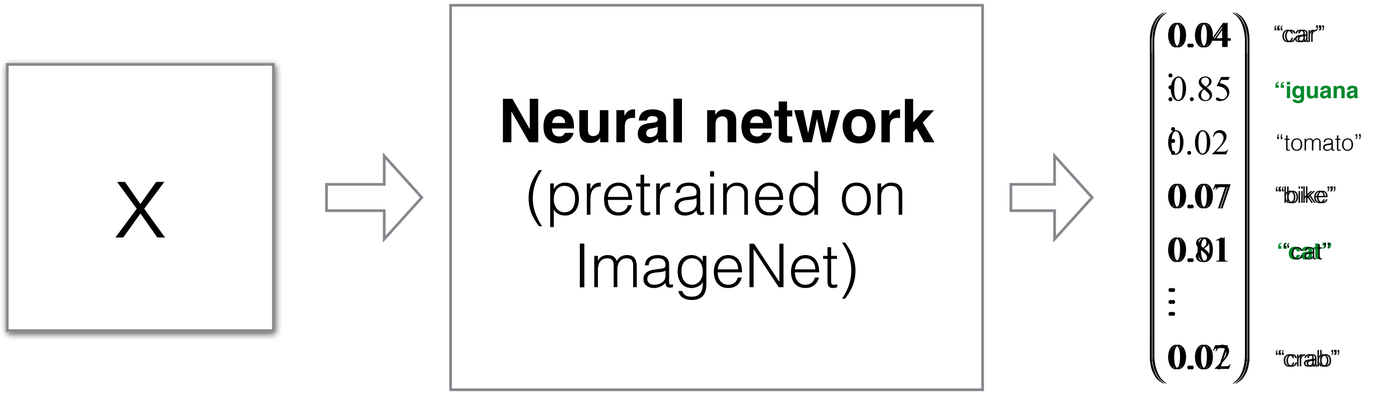
B. Defenses against adversarial examples

[Szegedy et al. (2013): Intriguing properties of neural networks]
[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

What are examples of Adversarial attacks?

I. A. Attacking a network with adversarial examples

Goal: Given a network pretrained on ImageNet, find an input image that will be classified as an iguana.



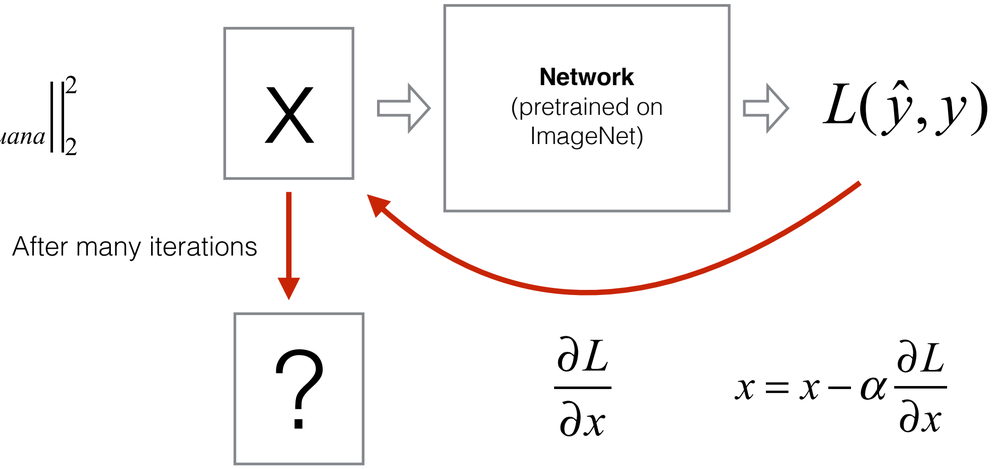
1. Rephrasing what we want:

Find x such that: $\hat{y}(x) = y_{iguana} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

2. Defining the loss function

$$L(\hat{y}, y) = \frac{1}{2}\left\| \hat{y}(W, b, x) - y_{iguana} \right\|_2^2$$

3. Optimize the image

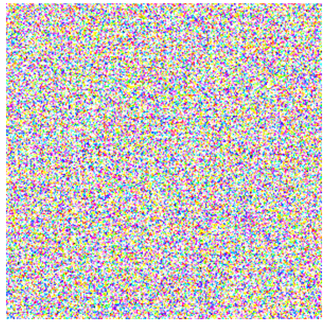


[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

Kian Katanforoosh

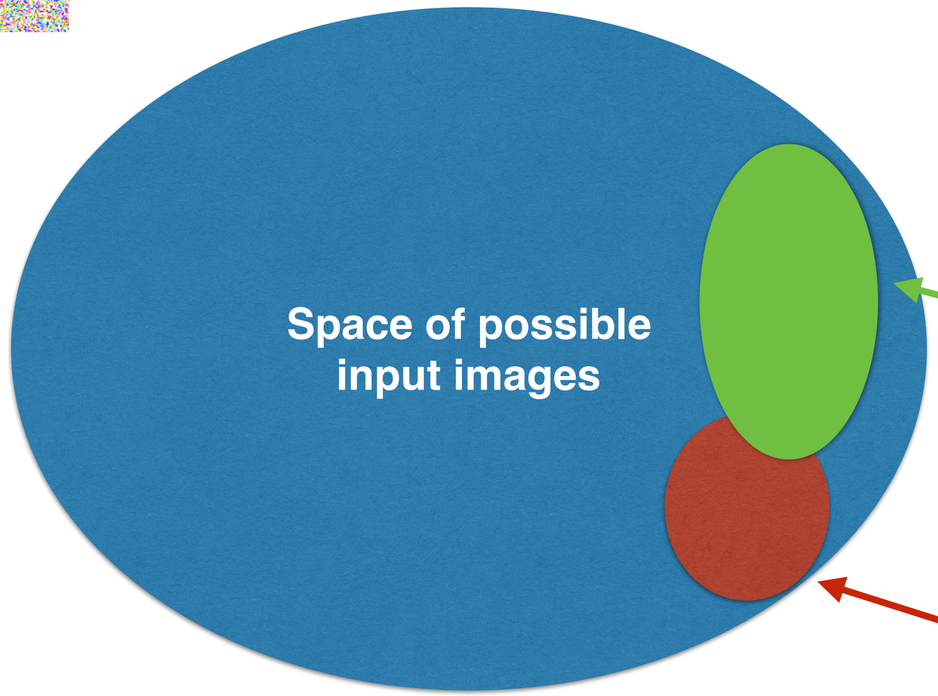
I. A. Attacking a network with adversarial examples

Question: Will the forged image **x** look like an **iguana**?



?

$$256^{32\times32\times3} \approx 10^{7400}$$



Kian Katanforoosh

I. A. Attacking a network with adversarial examples

Goal: Given a network pretrained on ImageNet, find an input image displaying a cat but classified as an iguana.

Neural network
(pretrained on ImageNet)

$$\begin{pmatrix} 0.04 \\ 0.85 \\ 0.02 \\ 0.07 \\ 0.81 \\ \vdots \\ 0.02 \end{pmatrix} \begin{matrix} \text{“car”} \\ \text{“iguana”} \\ \text{“tomato”} \\ \text{“bike”} \\ \text{“cat”} \\ \\ \text{“crab”} \end{matrix}$$

1. Rephrasing what we want:

Find x such that: $\hat{y}(x) = y_{iguana} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

And: $x = x_{cat}$

2. Defining the loss function

$$L(\hat{y}, y) = \frac{1}{2}\left\|\hat{y}(W,b,x) - y_{iguana}\right\|_2^2$$

$$+\lambda\left\|x - x_{cat}\right\|_2^2$$

3. Optimize the image

Network
(pretrained on ImageNet)

$L(\hat{y}, y)$

After many iterations

$$\frac{\partial L}{\partial x} \qquad x = x - \alpha \frac{\partial L}{\partial x}$$

[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

I. A. Attacking a network with adversarial examples

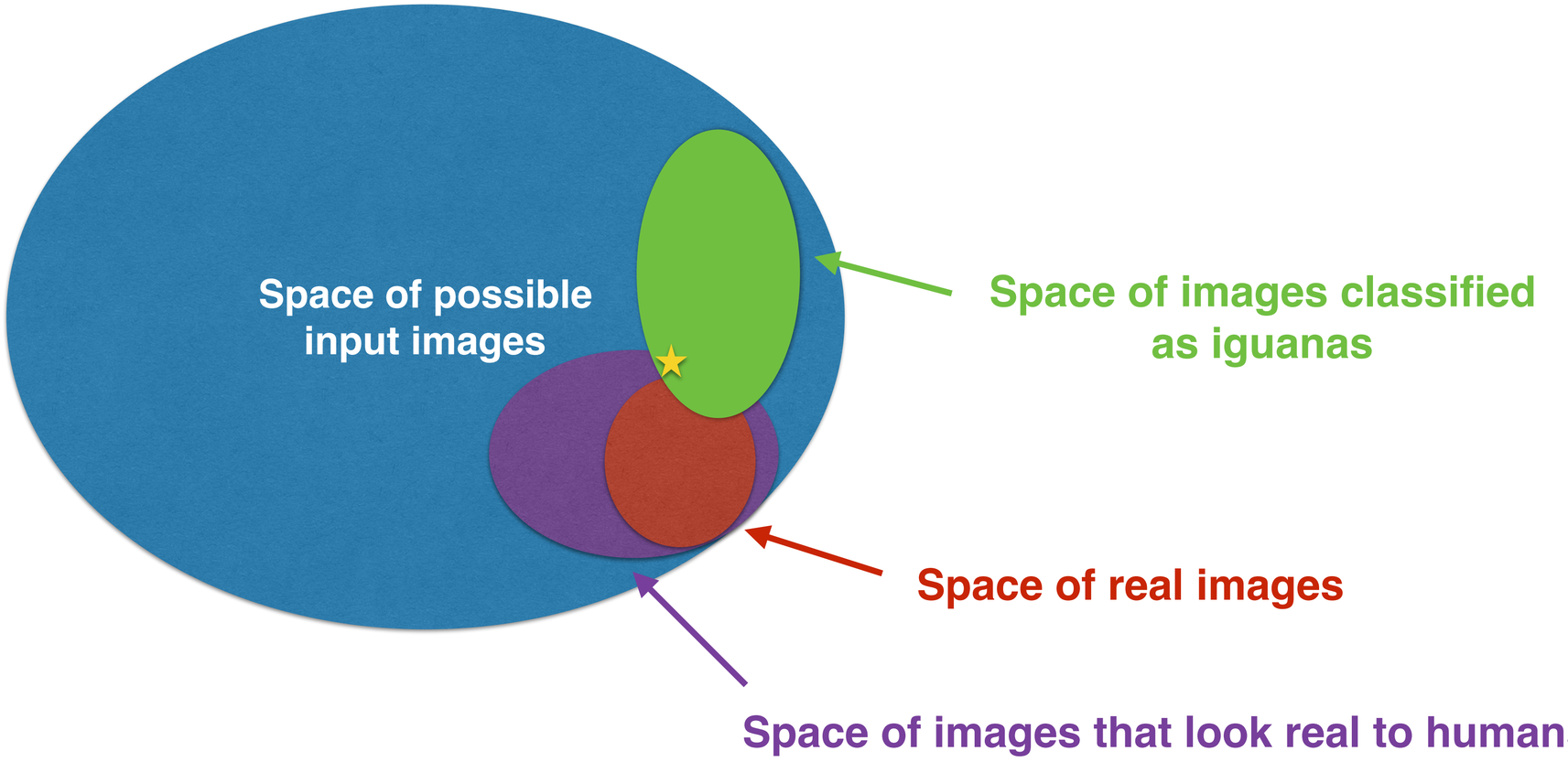
92% Cat

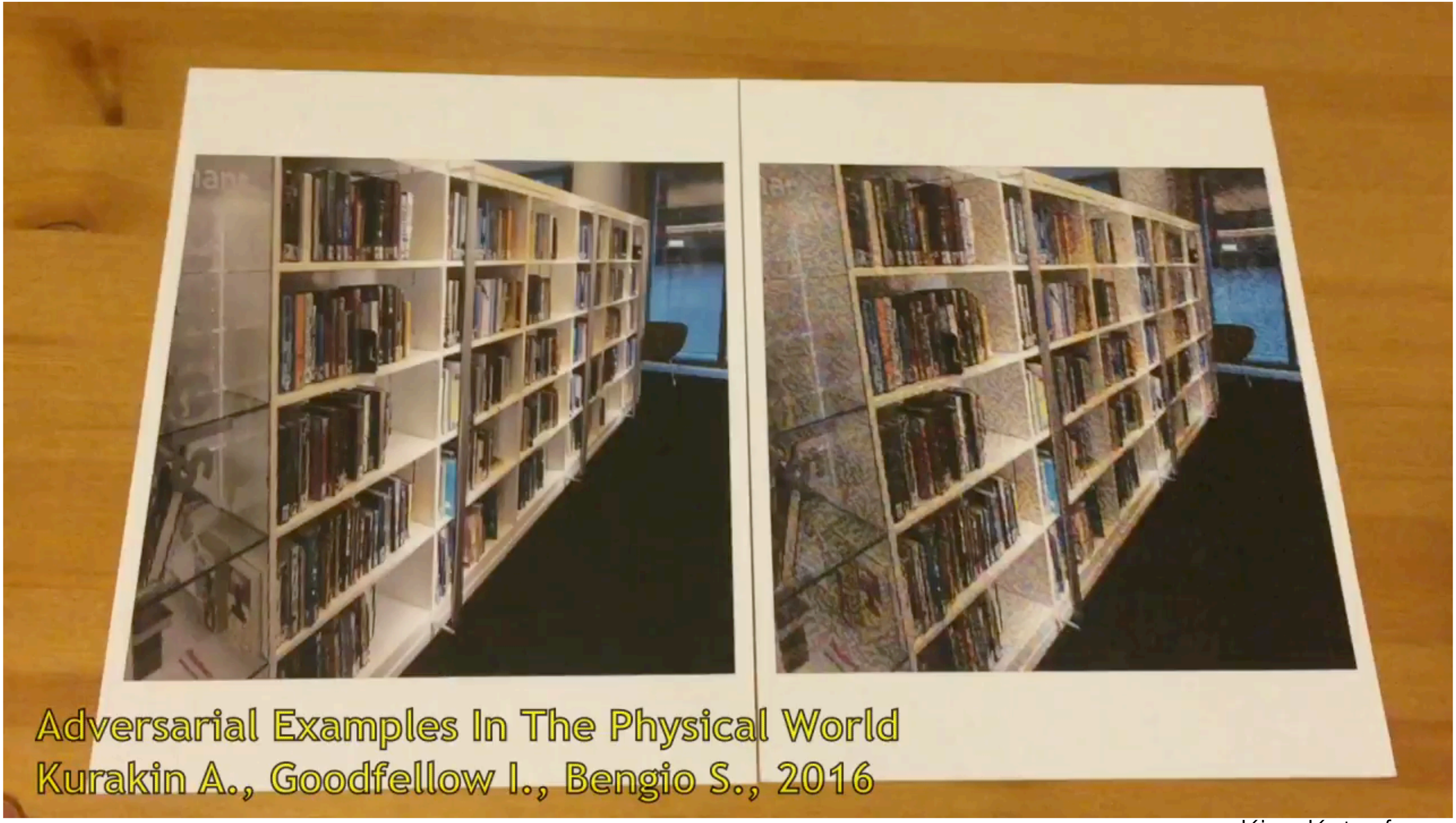
94% Iguana

Kian Katanforoosh

I. A. Attacking a network with adversarial examples

$$256^{32\times32\times3} \approx 10^{7400}$$





[Alexey Kurakin, Ian J. Goodfellow, Samy Bengio (2017): Adversarial examples in the physical world]

Kian Katanforoosh

I. C. Why are neural networks vulnerable to adversarial examples?

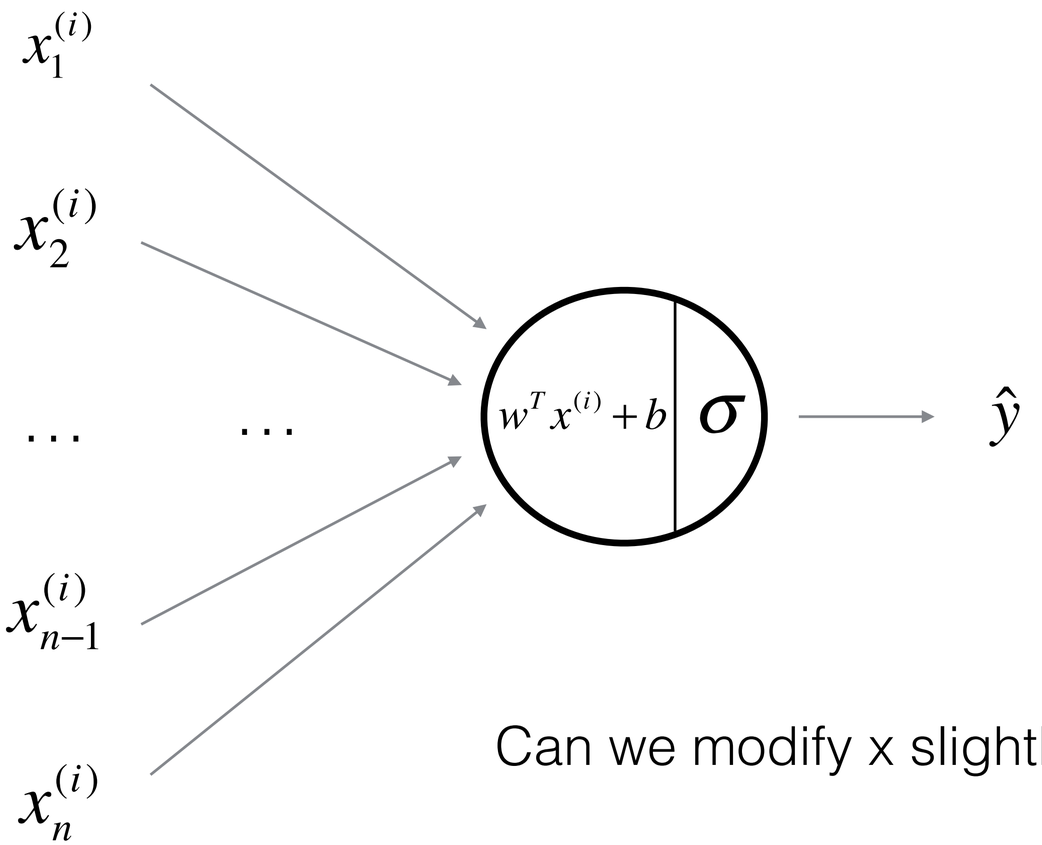
Get your pencils ready.

Do neural networks actually understand the data?

[Yuan et al. (2017): Adversarial Examples: Attacks and Defenses for Deep Learning]

I. C. Why are neural networks vulnerable to adversarial examples?

Let's design a method to generate Adversarial Examples



After successful training, we get:

$$w = (1,3,-1,2,2,3)^T$$
$$b = 0$$

For $x = (1,-1,2,0,3,-2)^T$

We get: $\hat{y} = 0.018$

Can we modify x slightly such that it affects $\hat{y}$ drastically?

$$x^* = x + \varepsilon w = x + 0.2w = (1.2,-0.4,1.8,0.4,3.4,-1.4)^T$$
$$\hat{y}(x^*) = \sigma(w^T x^*) = \sigma(w^T x + \varepsilon \mid w \mid^2) = 0.83$$

Kian Katanforoosh

I. C. Why are neural networks vulnerable to adversarial examples?

Fast Gradient Sign Method:

$$x^* = x + \varepsilon sign(\frac{dJ(W, X, Y)}{dX})$$

Table 3: Taxonomy of adversarial attacks covered in this paper.

Adversarial Attack(s)	Transparency	Specificity	Remarks
L-BFGS [31]	W	T, NT	Early attack on neural networks using constrained optimization method
FGSM [32]	W	T, NT	A fast single-step gradient ascent attack
BIM [45, 77]	W	T, NT	Iterative variants of FGSM
ILLCM [45, 77]	W	T	Extension of BIM to attack models with many output classes
R+FGSM [47]	W	T, NT	FGSM [32] with random initialization, can circumvent gradient masking
AMDR [78]	W	T, NT	Similar to L-BFGS but targetting feature space
DeepFool [79]	W	NT	Efficient method to find minimal perturbation that causes misclassification
JSMA [80]	W	T, NT	Some variants of JSMA can fool defensive distillation
SBA [41]	B	T, NT	Can fool defensive distillation [43], MagNet [81], gradient masking defenses
Hot/Cold [82]	W	T	Simultaneously moving towards "hot" class and away from "cold" class
C&W [44]	W	T, NT	Can fool defensive distillation [43], MagNet [81] and various detector networks
UAP [83]	W	NT	Generate input-agnostic perturbations
DFUAP [84]	W	NT	Generate input-agnostic perturbations without knowing any inputs
VAE Attacks [85]	W	T, NT	Can fool VAE [86] and potentially defenses relying on generative models
ATN [87]	W	T, NT	Generate adversarial examples using neural networks
DAG [88]	W	T, NT	Can fool semantic segmentation & object detection Models
ZOO [89]	B	T, NT	Can fool defensive distillation [43] and non-differentiable models
OPA [90]	B	T, NT	Uses genetic algorithm, can generate adversary by just modifying one pixel
Houdini [91]	W, B	T, NT	Method for attacking models directly through its non-differentiable metric
MI-FGSM [92]	W	T, NT	BIM + momentum, faster to converge and better transferability
AdvGAN [93]	W	T, NT	Generate adversarial examples using GAN [63]
Boundary Attack [94]	B	T, NT	Can fool defensive distillation [43] and non-differentiable models
NAA [60]	B	NT	Can generate adversaries for non-sensory inputs such as text
stAdv [95]	W	T, NT	Unique perceptual similarity objective
EOT [96]	W	T, NT	Good for creating physical adversaries and fooling randomization defenses
BPDA [55]	W	T, NT	Can fool various gradient masking defenses
SPSA [97]	B	T, NT	Can fool various gradient masking defenses
DDN [98]	W	T, NT	Better convergence compared to other constrained optimization methods
CAMOU [99]	B	NT	Attack in simulation using SBA [41], can be used to attack detection model

W: Whitebox
B: Blackbox
T: Targeted
NT: Non-targeted

[Wiyatno et al. (2019): Adversarial Examples in Modern Machine Learning: A Review]

I. B. Defenses against adversarial examples

Menti

Knowledge of the attacker:

- White-box
- Black-box

Examples of defenses (exploratory)

- Create a SafetyNet
- Train on correctly labelled adversarial examples
- Adversarial training $L_{new} = L(W, b, x, y) + \lambda L(W, b, x_{adv}, y)$

$x =$ 

y = cat

[Lu et al. (2017): SafetyNet: Detecting and Rejecting Adversarial Examples Robustly]
[Harini Kannan et al. (2018): Adversarial Logit Pairing]

Table 5: Taxonomy of adversarial defenses covered in this paper.

Adversarial Defense(s)	**Goal**	**Remarks**
Adversarial Training [32]	R	Training on adversarial examples
Ensemble Adversarial Training [47]	R	More robust to blackbox attacks compared to standard adversarial training
DCN [180]	R	Early defense against adversarial attacks with gradient regularization
Defensive Distillation [43]	R	Circumventable by C&W [44], SBA [41], and variant of JSMA [181]
MagNet [81]	R, D	Combination of R & D, circumventable by C&W [44] and SBA [41]
Random Resizing & Padding [51]	R	Circumventable by EOT variant (Expectation Over Randomness) [55]
SAP [50]	R	Circumventable by EOT variant (Expectation Over Randomness) [55]
TVM & Quilting [54]	R	Circumventable by combination of BPDA [55] and EOT [96]
TE [48]	R	Circumventable by BPDA [55]
PixelDefend [53]	R, D	Circumventable by BPDA [55] and SPSA [97]
Defense-GAN [52]	R	Circumventable by BPDA [55]
PGD Adversarial Training [145]	R	Training only on PGD adversaries
WRM [182]	R	Adversarial training with robustness certificate
HGD [159]	R	Circumventable by SPSA [97]
ALP [183]	R	Circumventable by PGD [145] with many iterations [184]
FN [185]	R	Denoising on hidden representations using autoencoders
FDB [186]	R	Denoising on hidden representations using differentiable denoising operation
ABS [187]	R	Model distribution of the inputs for each class using VAE [86]
WSNNS [188]	R	Replace input with its nearest neighbor from a large database of images
ME-Net [189]	R	Defense using matrix estimation algorithms
H&G's Methods [190, 191]	D	Circumventable by modified C&W [44]
Detector Networks [192, 193, 194, 195]	D	Circumventable by C&W [44] and SBA [41]
KDE & BUE [196]	D	Circumventable by modified C&W [44]
Feature Squeezing [197]	D	Detection by comparing the predictions between preprocessed and original inputs
RCE [198]	D	Defense using reverse crossentropy loss

R: Robustness
D: Detection

[Wiyatno et al. (2019): Adversarial Examples in Modern Machine Learning: A Review]

II. Generative Adversarial Networks (GANs)

A. Motivation

B. G/D Game

C. Training GANs

D. Nice results

E. In terms of code

[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

II.A - Motivation

Motivation:
- Data synthesis
- Compress and reconstruct data.
- Find a mapping between spaces.
- Image in-painting

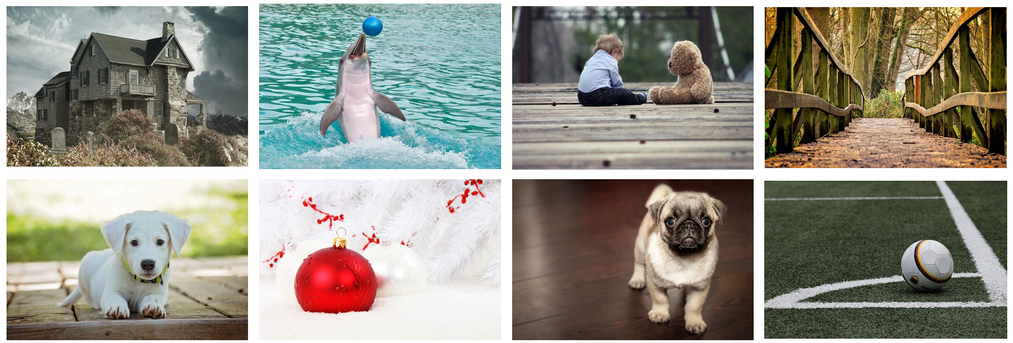
Approach: Collect a lot of data, use it to train a model to generate similar data from scratch.

Intuition: number of parameters of the model << amount of data

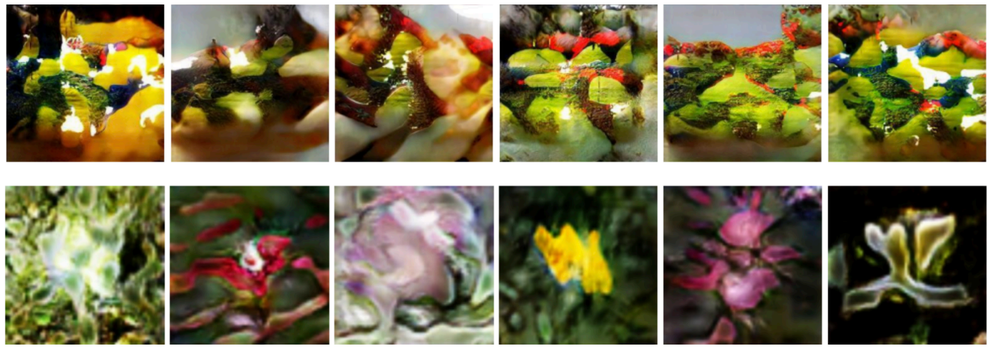
II.A - Motivation

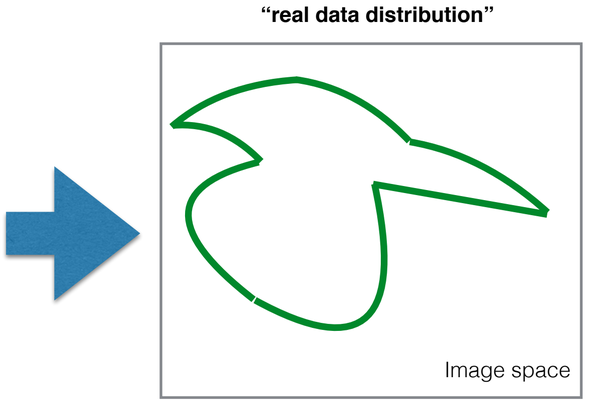
Probability distributions:

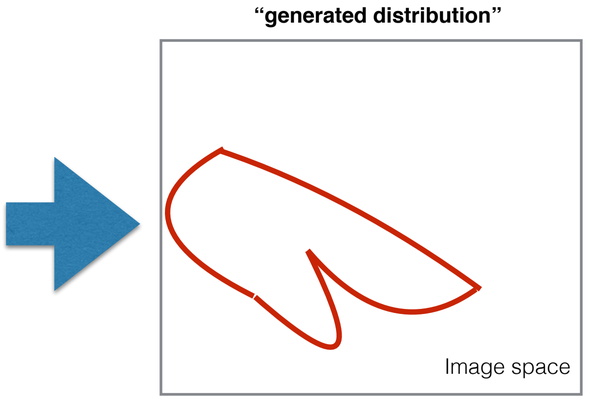
Samples from the “real data distribution”



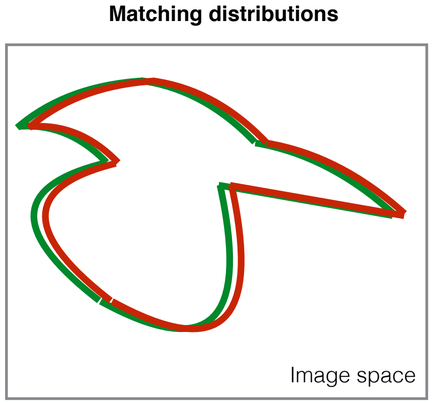
Samples from the “generated distribution”







Goal



[Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas (2017): StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks]

II. Generative Adversarial Networks (GANs)

A. Motivation

B. G/D Game

C. Training GANs

D. Nice results

E. In terms of code

[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

II.B - G/D Game

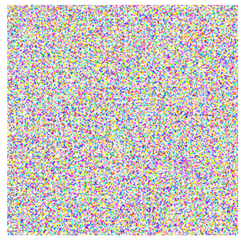
100-d
random code

$$\begin{pmatrix} 0.47 \\ \vdots \\ 0.19 \end{pmatrix}$$

z

**Generator "G"
(Neural Network)**

(64,64,3)
generated image



$\neq$

How can we train G to generate images from the true data distributions?

[Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas (2017): StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks]

II.B - G/D Game

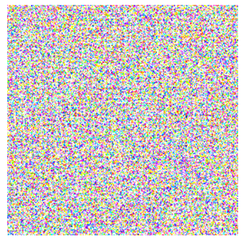
100-d
random code

$$\begin{pmatrix} 0.47 \\ \vdots \\ 0.19 \end{pmatrix}$$

z

**Generator "G"
(Neural Network)**

(64,64,3)
generated image



Run Gradient Descent simultaneously on two minibatches (true data / generated data)

x

**Discriminator "D"
(Neural Network)**

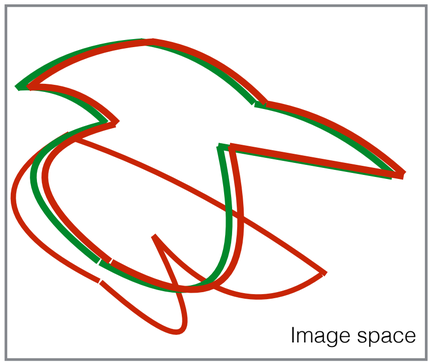
Gradients

Binary classification

$$\begin{cases} y = 0 \quad if \quad x = G(z) \\ y = 1 \quad otherwise \end{cases}$$

**Real images
(database)**

Probability distributions



Kian Katanforoosh

II.B - G/D Game

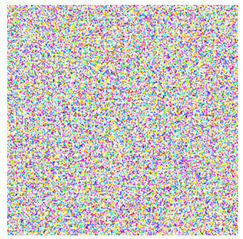
100-d
random code

$$\begin{pmatrix} 0.47 \\ \vdots \\ 0.19 \end{pmatrix}$$

z

**Generator "G"
(Neural Network)**

(64,64,3)
generated image



x

**Discriminator "D"
(Neural Network)**

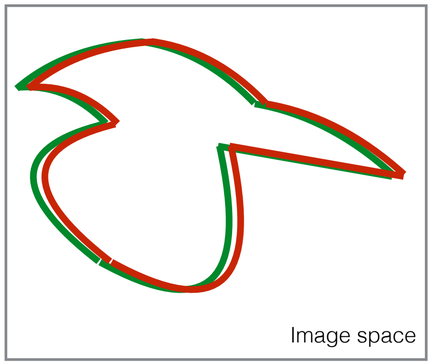
Gradients Binary classification

$$\begin{cases} y = 0 \quad if \quad x = G(z) \\ y = 1 \quad otherwise \end{cases}$$

End goal: G is outputting images that are indistinguishable from real images for D

**Real images
(database)**

Probability distribution



Kian Katanforoosh

II.B - G/D Game

Training procedure, we want to minimize:

Labels: $\begin{cases} y_{real} & \text{is always 1} \\ y_{gen} & \text{is always 0} \end{cases}$

- The cost of the discriminator

$$J^{(D)} = \underbrace{-\frac{1}{m_{real}} \sum_{i=1}^{m_{real}} y_{real}^{(i)} . \log(D(x^{(i)}))}_{\text{cross-entropy 1}} \underbrace{- \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} (1 - y_{gen}^{(i)}) . \log(1 - D(G(z^{(i)})))}_{\text{cross-entropy 2}}$$

cross-entropy 1:
"D should correctly label real data as 1"

cross-entropy 2:
"D should correctly label generated data as 0"

- The cost of the generator

$$J^{(G)} = -J^{(D)} = \frac{1}{m_{gen}} \sum_{i=1}^{m_{gen}} \log(1 - D(G(z^{(i)})))$$

"G should try to fool D: by minimizing the opposite of what D is trying to minimize"

II. Generative Adversarial Networks (GANs)

A. Motivation

B. G/D Game

C. Training GANs

D. Nice results

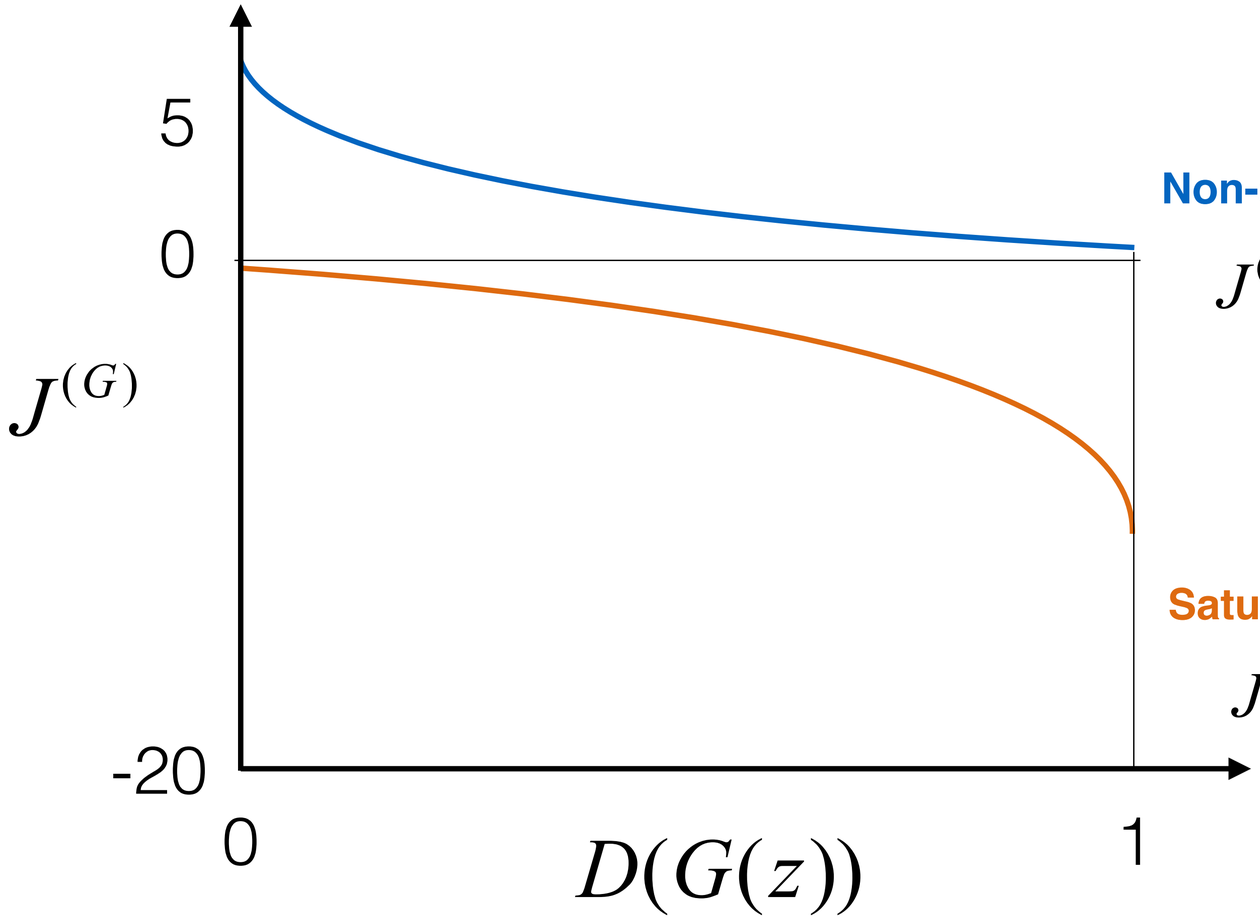
E. In terms of code

[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

II.C - Training GANs

Saturating cost for the generator:

$$\min\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D(G(z^{(i)})))\right] \Leftrightarrow \max\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right] \Leftrightarrow \min\left[-\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right]$$



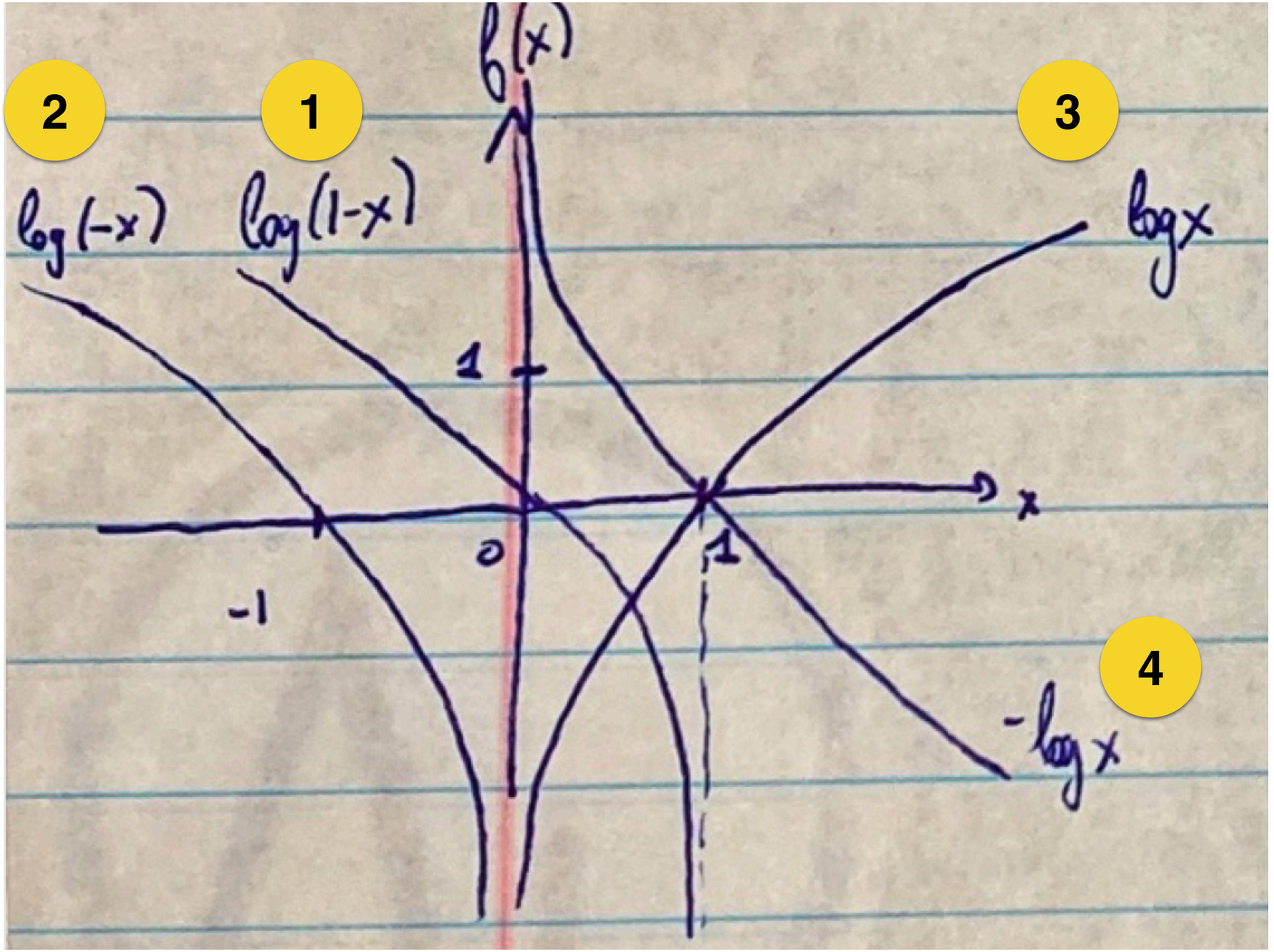
Non-saturating cost

$$J^{(G)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))$$

Saturating cost

$$J^{(G)} = \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D(G(z^{(i)})))$$

[Ian Goodfellow (2014): NIPS Tutorial: GANs]



Kian Katanforoosh

II.C - Training GANs

Note that: $$\min\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D(G(z^{(i)})))\right] \Leftrightarrow \max\left[\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right] \Leftrightarrow \min\left[-\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))\right]$$

New training procedure, we want to minimize:

$$J^{(D)} = \underbrace{-\frac{1}{m_{real}}\sum_{i=1}^{m_{real}} y_{real}^{(i)}.\log(D(x^{(i)}))}_{\text{cross-entropy 1:}} \underbrace{-\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}(1-y_{gen}^{(i)}).\log(1-D(G(z^{(i)})))}_{\text{cross-entropy 2:}}$$

cross-entropy 1:
"D should correctly label real data as 1"

cross-entropy 2:
"D should correctly label generated data as 0"

$$J^{(G)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D(G(z^{(i)})))$$

"G should try to fool D: by minimizing this"

Table 1: Generator and discriminator loss functions. The main difference whether the discriminator outputs a probability (MM GAN, NS GAN, DRAGAN) or its output is unbounded (WGAN, WGAN GP, LS GAN, BEGAN), whether the gradient penalty is present (WGAN GP, DRAGAN) and where is it evaluated. We chose those models based on their popularity.

GAN	DISCRIMINATOR LOSS	GENERATOR LOSS
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1)^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x-\mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$

[Lucic, Kurach et al. (2018): Are GANs Created Equal? A Large-Scale Study]

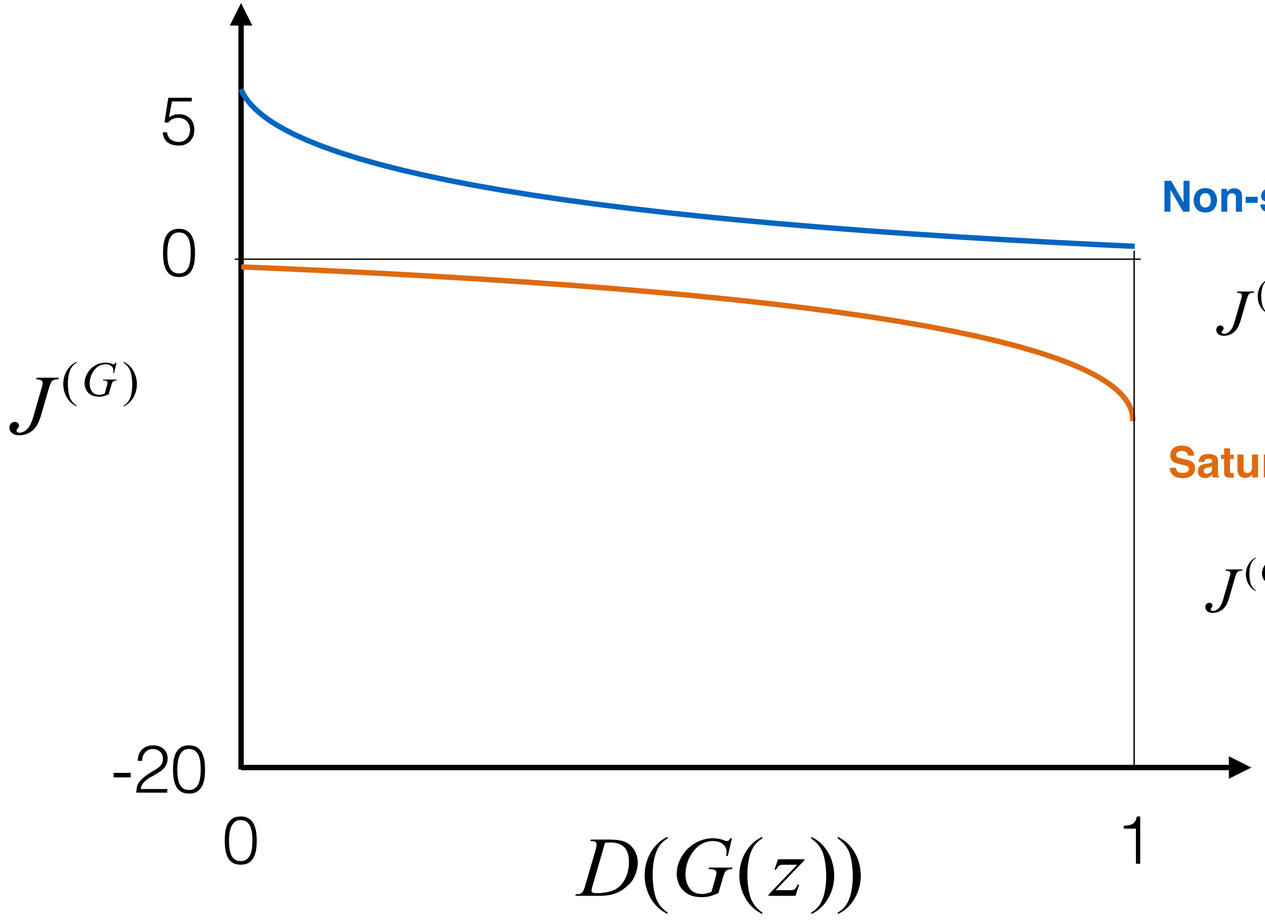
II.C - Training GANs

Simultaneously training G/D?

for num_iterations:

for k iterations:
update D

update G



Non-saturating cost

$$J^{(G)} = -\frac{1}{m_g}\sum_{i=1}^{m_g}\log(D(G(z^{(i)})))$$

Saturating cost

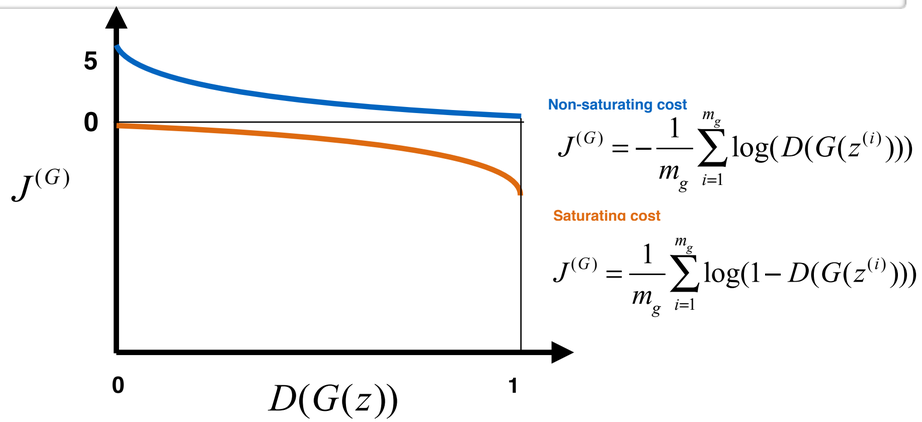
$$J^{(G)} = \frac{1}{m_g}\sum_{i=1}^{m_g}\log(1-D(G(z^{(i)})))$$

[Ian Goodfellow (2014): NIPS Tutorial: GANs]

II.C - Training GANs

Recap: GANs' training tips

- Modification of the cost function



- Keep D up-to-date with respect to G (k update for D / 1 update for G)

And a lot more, GANs are hard to train!

[Soumith et al. (2016): GanHacks]
[Lucic, Kurach et al. (2018): Are GANs Created Equal? A Large-Scale Study]

II. Generative Adversarial Networks (GANs)

A. Motivation

B. G/D Game

C. Training GANs

D. Nice results

E. In terms of code

[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

II.E - Nice results

Operation on codes

Code 1

$\begin{pmatrix} 0.12 \\ \vdots \\ 0.92 \end{pmatrix}$

Generator "G" (Neural Network)

(64,64,3) generated image

Code 2

$\begin{pmatrix} 0.47 \\ \vdots \\ 0.19 \end{pmatrix}$

Generator "G" (Neural Network)

(64,64,3) generated image

Code 3

$\begin{pmatrix} 0.42 \\ \vdots \\ 0.07 \end{pmatrix}$

Generator "G" (Neural Network)

(64,64,3) generated image

Code 1 Code 2 Code 3

$\begin{pmatrix} 0.12 \\ \vdots \\ 0.92 \end{pmatrix} - \begin{pmatrix} 0.47 \\ \vdots \\ 0.19 \end{pmatrix} + \begin{pmatrix} 0.42 \\ \vdots \\ 0.07 \end{pmatrix}$

Generator "G" (Neural Network)

Man with glasses - man + woman = woman with glasses

[Radford et al. (2015): UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS]

II.E - Nice results

Face Generation:

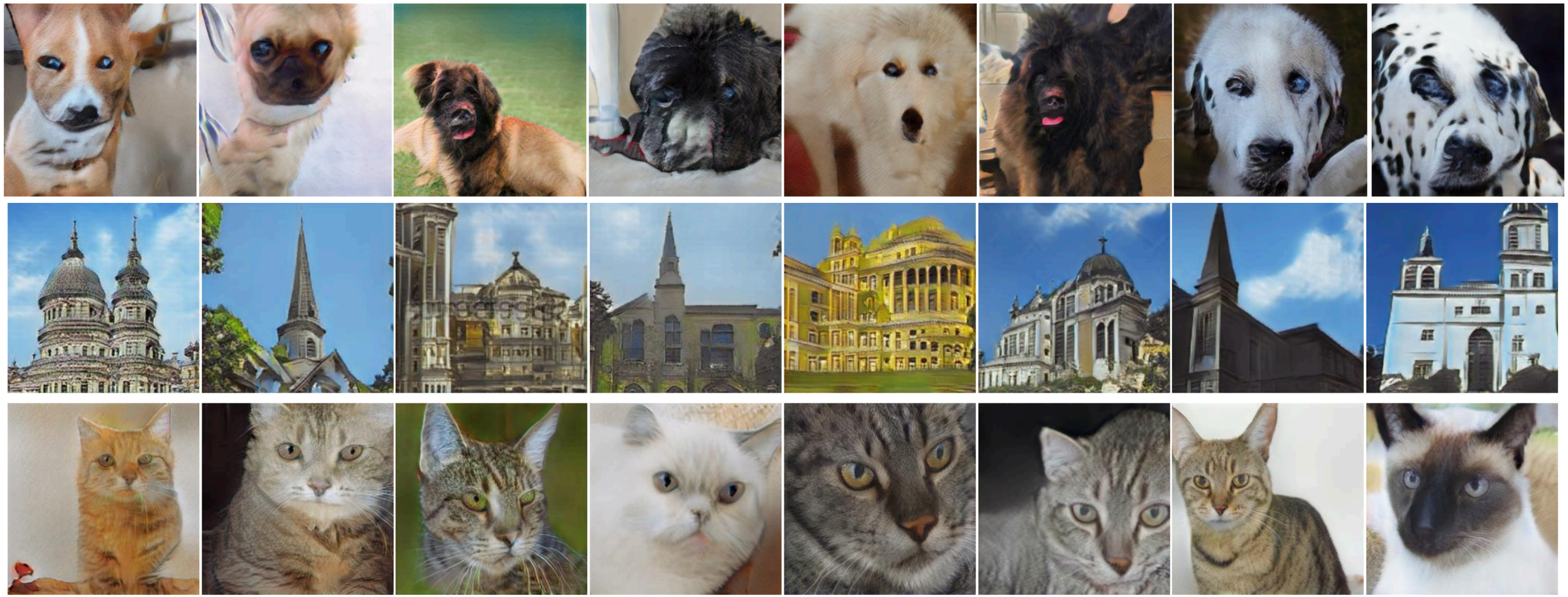
[Karras et al. (2018): A Style-Based Generator Architecture for Generative Adversarial Networks]

https://www.youtube.com/watch?v=kSLJriaOumA&feature=youtu.be

II.E - Nice results

Image Generation:

Samples from the “generated distribution”



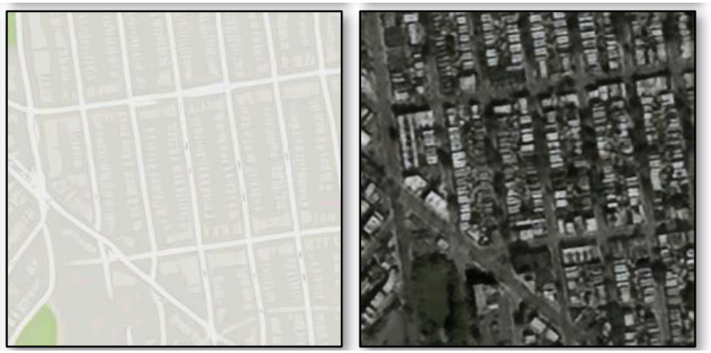
[Zhang et al. (2017): StackGAN++]

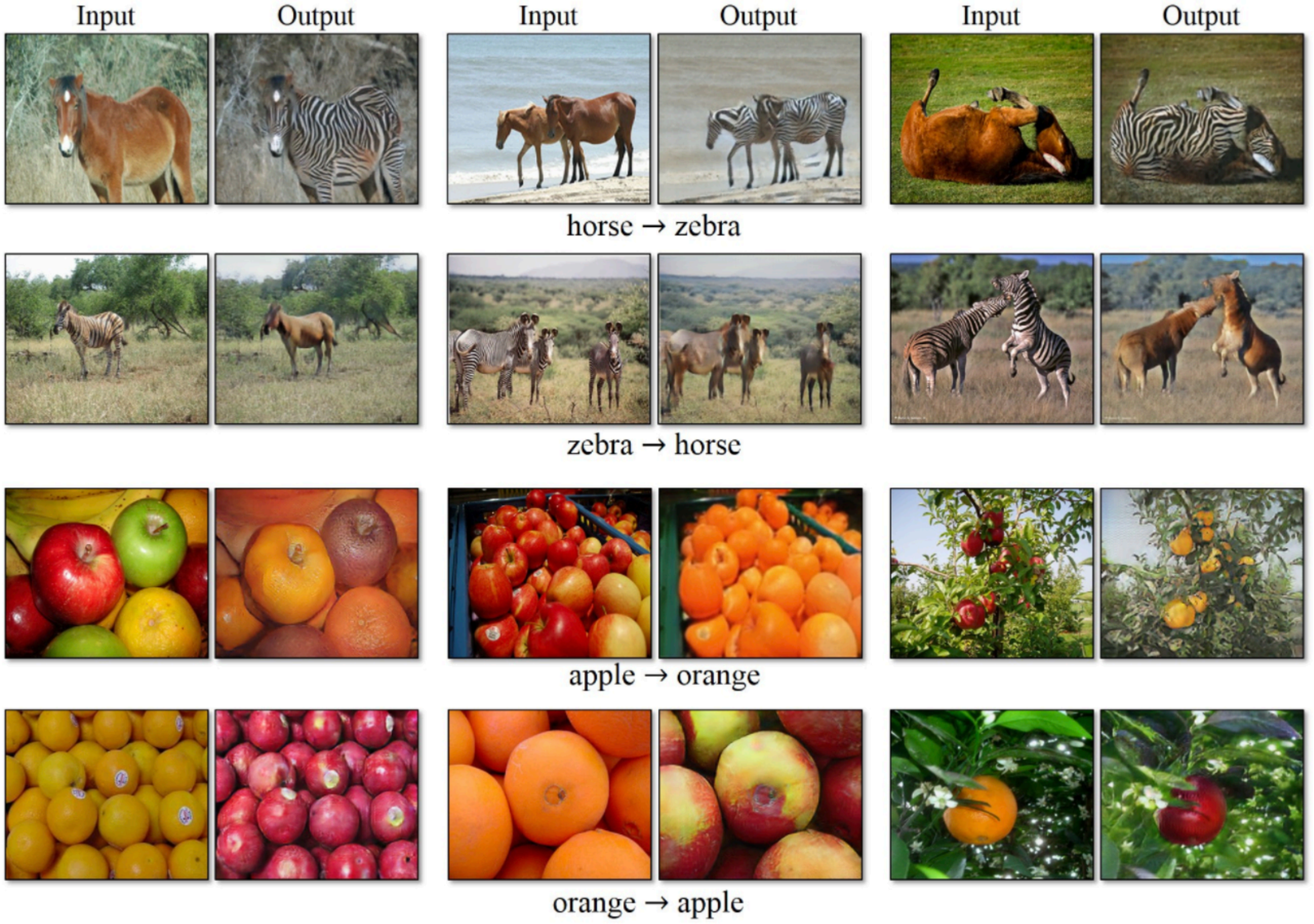
II.E - Nice results

Figure 3: Street scene image translation results. For each pair, left is input and right is the translated image.

[Liu et al. (2017): Unsupervised Image-to-Image Translation Networks]

II.E - Nice results





[Zhu, Park et al. (2017): Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks]

II.E - Nice results

Goal: Convert horses to zebras on images, and vice-versa.

Data? **Architecture?** **Cost function?**

Unpaired images

Horse images Zebra images

[Zhu, Park et al. (2017): Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks]

II.E - Nice results

Architecture?

$$\begin{cases} y = 0 \quad if \quad x = G1(H) \\ y = 1 \quad otherwise \quad (x = z) \end{cases}$$

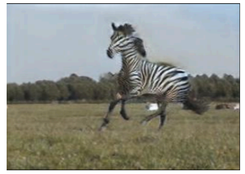
H2Z

$$\begin{cases} y = 0 \quad if \quad x = G2(Z) \\ y = 1 \quad otherwise \quad (x = h) \end{cases}$$

H

Generator1 (H2Z)

G1(H)



Discriminator1

G2(G1(H))

Generator2 (Z2H)

Discriminator2

$$\begin{cases} y = 0 \quad if \quad x = G1(H) \\ y = 1 \quad otherwise \quad (x = z) \end{cases}$$

G1(G2(Z))

Generator1 (H2Z)

Discriminator1

$$\begin{cases} y = 0 \quad if \quad x = G2(Z) \\ y = 1 \quad otherwise \quad (x = h) \end{cases}$$

G2(Z)

Generator2 (Z2H)

Z

Discriminator2

Z2H

[Zhu, Park et al. (2017): Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks]

Kian Katanforoosh

II.E - Nice results

Loss to minimize?

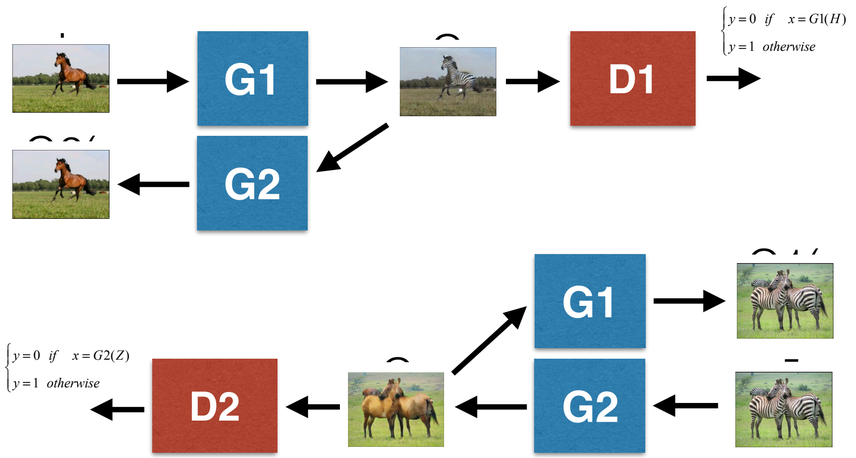
$$J^{(D1)} = -\frac{1}{m_{real}}\sum_{i=1}^{m_{real}}\log(D1(z^{(i)})) - \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D1(G1(H^{(i)})))$$

$$J^{(G1)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D1(G1(H^{(i)})))$$

$$J^{(D2)} = -\frac{1}{m_{real}}\sum_{i=1}^{m_{real}}\log(D2(h^{(i)})) - \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D2(G2(Z^{(i)})))$$

$$J^{(G2)} = -\frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(D2(G2(Z^{(i)})))$$

$$J^{cycle} = \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\| G2(G1(H^{(i)}) - H^{(i)} \|_1 + \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\| G1(G2(Z^{(i)}) - Z^{(i)} \|_1$$



$$J = J^{(D1)} + J^{(G1)} + J^{(D2)} + J^{(G2)} + \lambda J^{cycle}$$

Kian Katanforoosh

II.E - Nice results

CycleGANs:

Face2ramen

+ Face detection

[Shu Naritomi et al.: Face2Ramen]
[Takuya Tako: Face2Ramen using CycleGAN]
[Zhu, Park et al. (2017): Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks]

II.E - Nice results

Pix2Pix:

https://affinelayer.com/pixsrv/ by Christopher Hesse.

[Isola et al. (2017): Image-to-Image Translation with Conditional Adversarial Networks]

II.E - Nice results

Super-resolution

CS230

Human Portrait Super Resolution Using GANs

Yujie Shu
[redacted]

Figure 1: Input LR 32x32, SRPGGAN 8x Output 256x256, and Original HR 256x256

[Ledig et al. (2016): Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network]

II.E - Nice results

Motion Retargeting video subjects : https://www.youtube.com/watch?

[Chan et al. (2018): Everybody Dance Now]

II.E - Nice results

Other applications of GANs:

- Beaulieu-Jones et al., Privacy-preserving generative deep neural networks support clinical data sharing.
- Hwang et al., Learning Beyond Human Expertise with Generative Models for Dental Restorations.
- Gomez et al., Unsupervised cipher cracking using discrete GANs.
- Many more…

Announcements

For next week:

Completed modules (due before lecture):

- C2M1: Practical aspects of deep learning (slides)
- C2M2: Optimization algorithms (slides)

Quizzes (due before lecture):

- Practical aspects of deep learning
- Optimization Algorithms

Programming Assignments (due before lecture):

- Initialization
- Regularization
- Gradient Checking
- Optimization

This Friday: TA section

II. Generative Adversarial Networks (GANs)

A. Motivation

B. G/D Game

C. Training GANs

D. Nice results

E. In terms of code

[Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy (2015): Explaining and harnessing adversarial examples]

II. D. In terms of code

```python
# Build and compile the discriminator
self.discriminator = self.build_discriminator()
self.discriminator.compile(loss='binary_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy'])

# Build the generator
self.generator = self.build_generator()

# The generator takes noise as input and generates imgs
z = Input(shape=(self.latent_dim,))
img = self.generator(z)

# For the combined model we will only train the generator
self.discriminator.trainable = False

# The discriminator takes generated images as input and determines validity
validity = self.discriminator(img)

# The combined model  (stacked generator and discriminator)
# Trains the generator to fool the discriminator
self.combined = Model(z, validity)
self.combined.compile(loss='binary_crossentropy', optimizer=optimizer)
```

```python
def build_discriminator(self):

    model = Sequential()

    model.add(Flatten(input_shape=self.img_shape))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()

    img = Input(shape=self.img_shape)
    validity = model(img)

    return Model(img, validity)
```

[Erik Linder-Norén (Github): eriklindernoren/Keras-GAN: link]

II. D. In terms of code

```
# Build and compile the discriminator
self.discriminator = self.build_discriminator()
self.discriminator.compile(loss='binary_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy'])

# Build the generator
self.generator = self.build_generator()

# The generator takes noise as input and generates imgs
z = Input(shape=(self.latent_dim,))
img = self.generator(z)

# For the combined model we will only train the generator
self.discriminator.trainable = False

# The discriminator takes generated images as input and determines validity
validity = self.discriminator(img)

# The combined model  (stacked generator and discriminator)
# Trains the generator to fool the discriminator
self.combined = Model(z, validity)
self.combined.compile(loss='binary_crossentropy', optimizer=optimizer)
```

```
def build_generator(self):

    model = Sequential()

    model.add(Dense(256, input_dim=self.latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(1024))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(np.prod(self.img_shape), activation='tanh'))
    model.add(Reshape(self.img_shape))

    model.summary()

    noise = Input(shape=(self.latent_dim,))
    img = model(noise)

    return Model(noise, img)
```

[Erik Linder-Norén (Github): eriklindernoren/Keras-GAN: link]

II. D. In terms of code

```python
def train(self, epochs, batch_size=128, sample_interval=50):

    # Load the dataset
    (X_train, _), (_, _) = mnist.load_data()

    # Rescale -1 to 1
    X_train = X_train / 127.5 - 1.
    X_train = np.expand_dims(X_train, axis=3)

    # Adversarial ground truths
    valid = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    for epoch in range(epochs):

        # ---------------------
        #  Train Discriminator
        # ---------------------

        # Select a random batch of images
        idx = np.random.randint(0, X_train.shape[0], batch_size)
        imgs = X_train[idx]

        noise = np.random.normal(0, 1, (batch_size, self.latent_dim))

        # Generate a batch of new images
        gen_imgs = self.generator.predict(noise)

        # Train the discriminator
        d_loss_real = self.discriminator.train_on_batch(imgs, valid)
        d_loss_fake = self.discriminator.train_on_batch(gen_imgs, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # ---------------------
        #  Train Generator
        # ---------------------

        noise = np.random.normal(0, 1, (batch_size, self.latent_dim))

        # Train the generator (to have the discriminator label samples as valid)
        g_loss = self.combined.train_on_batch(noise, valid)
```

[Erik Linder-Norén (Github): eriklindernoren/Keras-GAN: link]