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

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

A. Attacking a network with adversarial examples

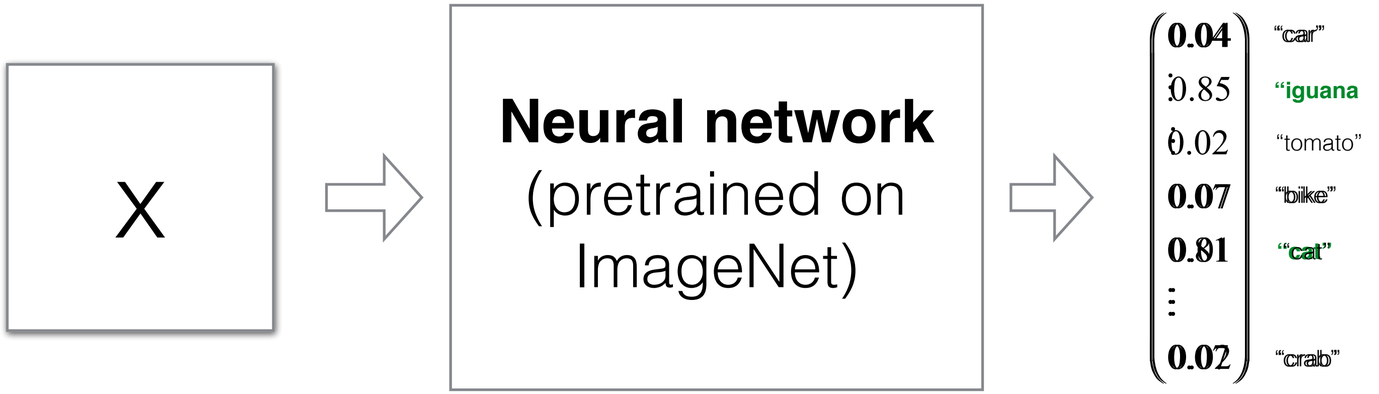
B. Defenses against adversarial examples

C. Why are neural networks vulnerable to adversarial examples?

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

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.



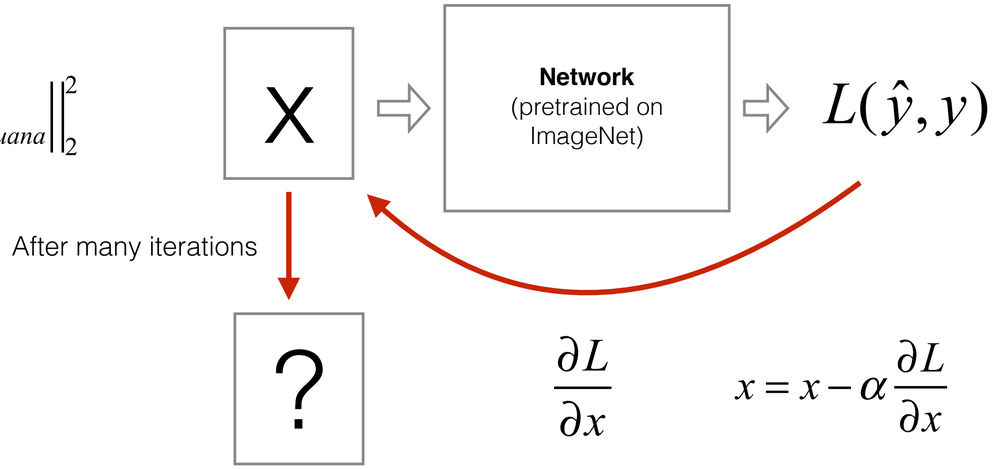
1. Rephrasing what we want:

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

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

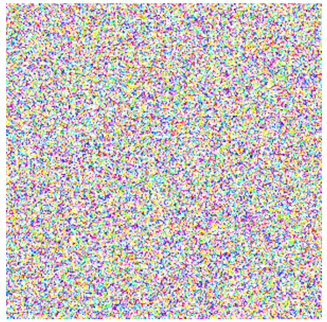
3. Optimize the image



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

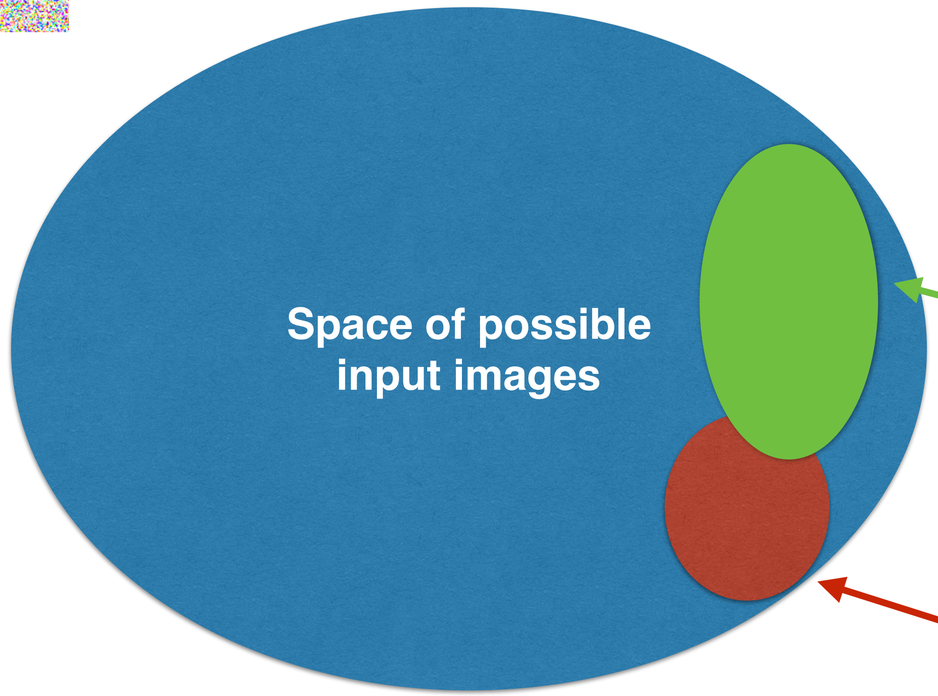
I. A. Attacking a network with adversarial examples

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

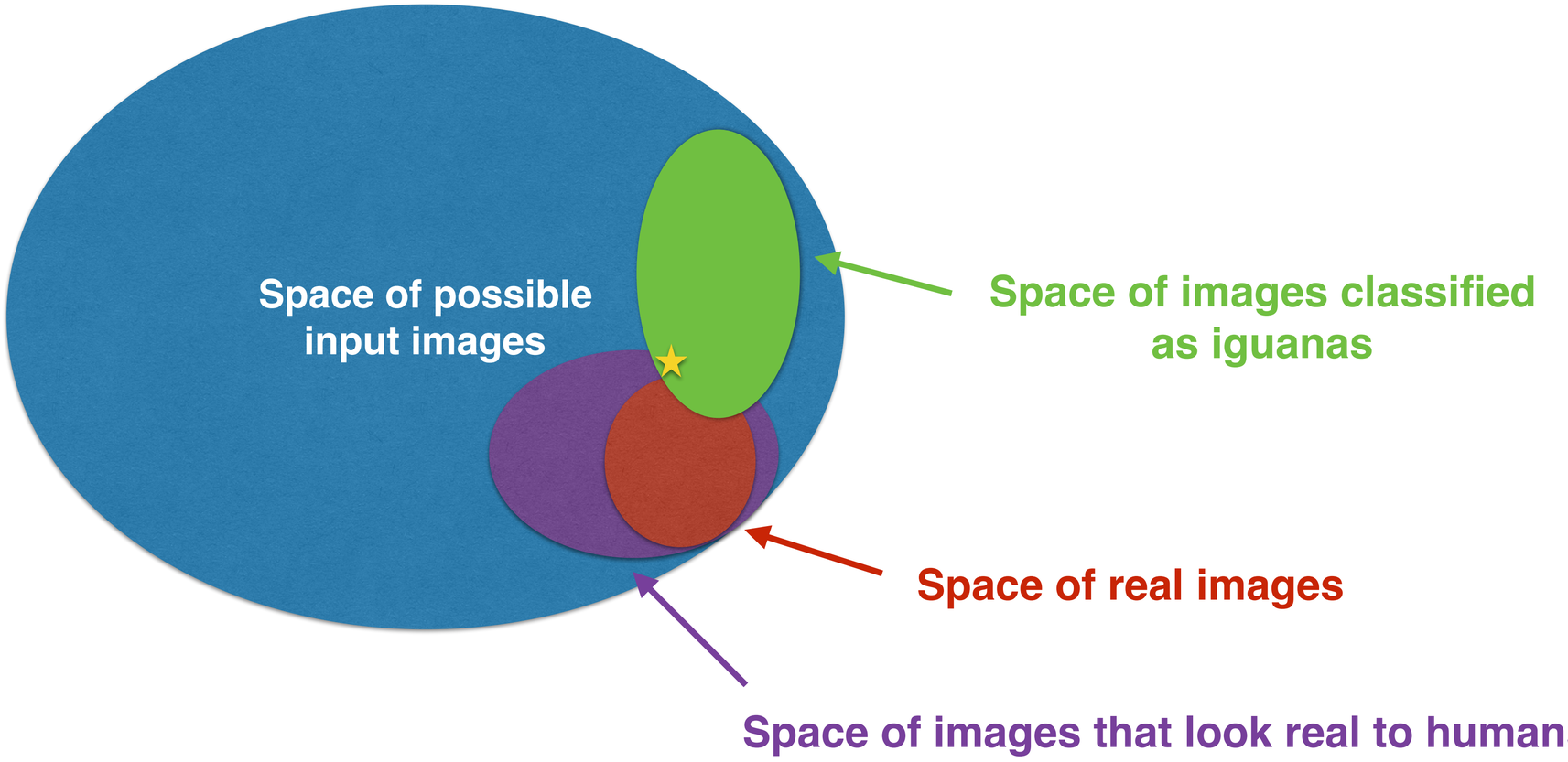
92% Cat

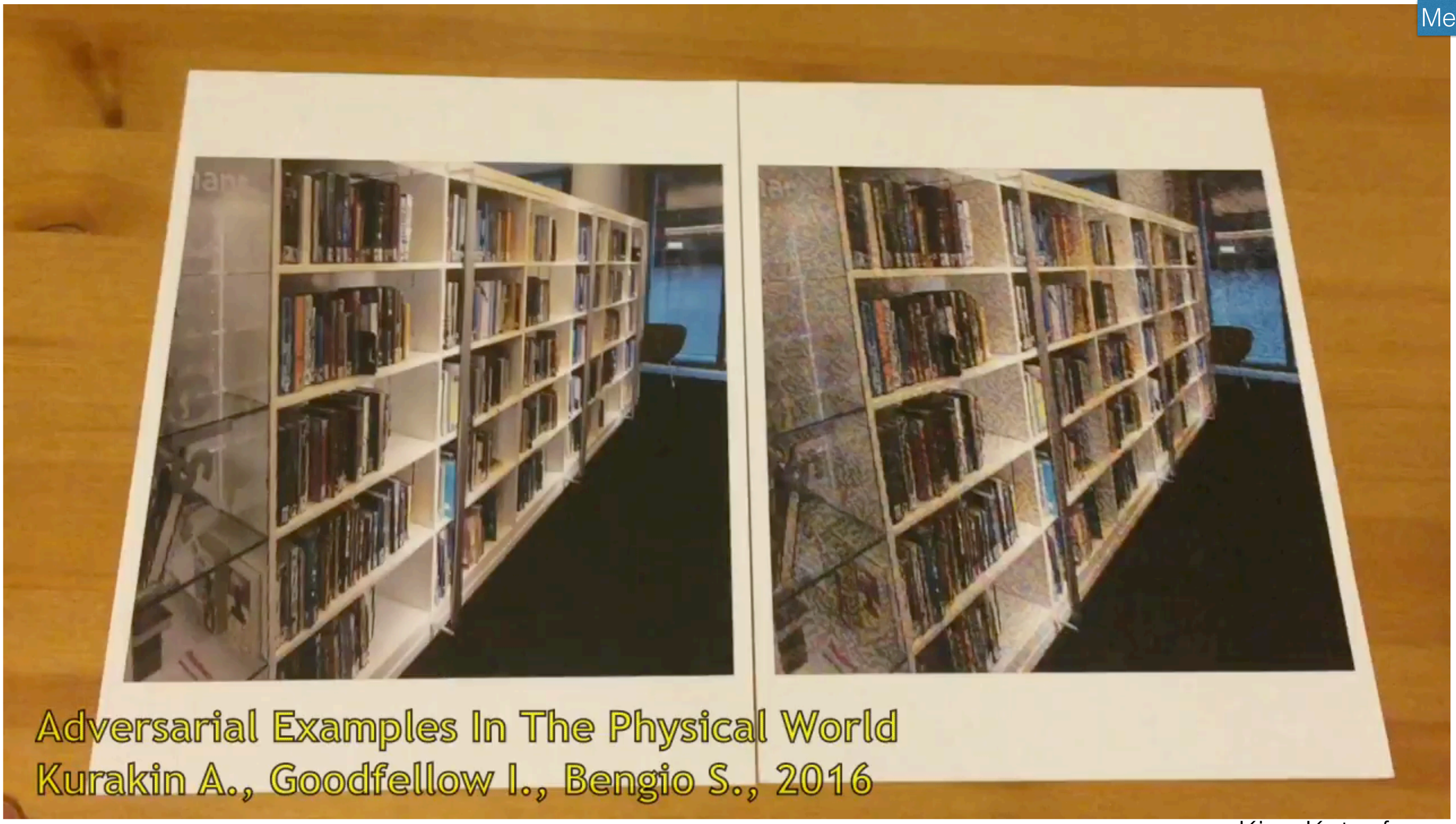
94% Iguana

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

Menti

I. B. Defenses against adversarial examples

Menti

Knowledge of the attacker:

- White-box
- Black-box

Solution 1

- Create a SafetyNet

$x =$ 

$y =$ cat

Solution 2

- Train on correctly labelled adversarial examples

Solution 3

- Adversarial training $L_{new} = L(W, b, x, y) + \lambda L(W, b, x_{adv}, y)$

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

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

Get your pencils ready, we're switching to iPad.

Do neural networks actually understand the data?

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

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

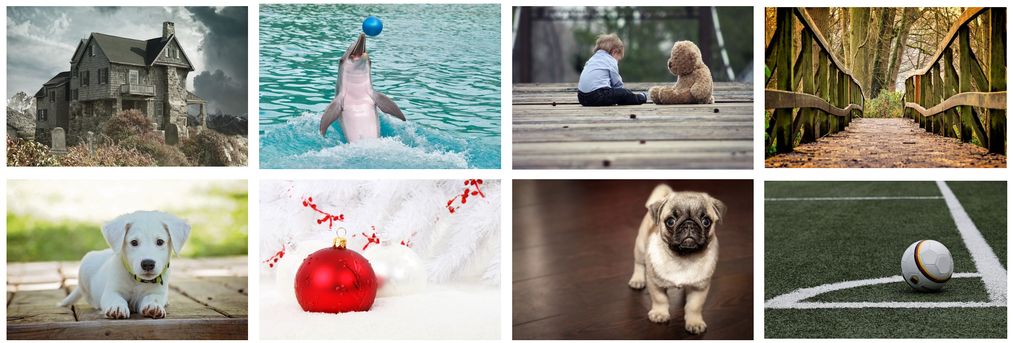
Approach: Collect a lot of data, use it to train a model to generate similar data from scratch.

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

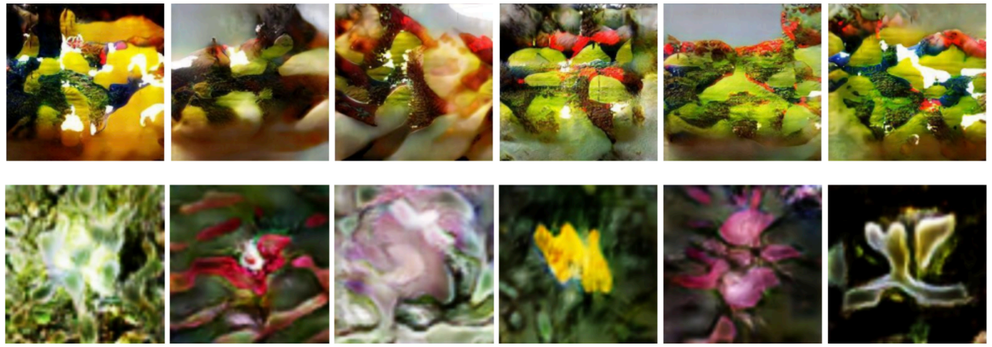
II.A - Motivation

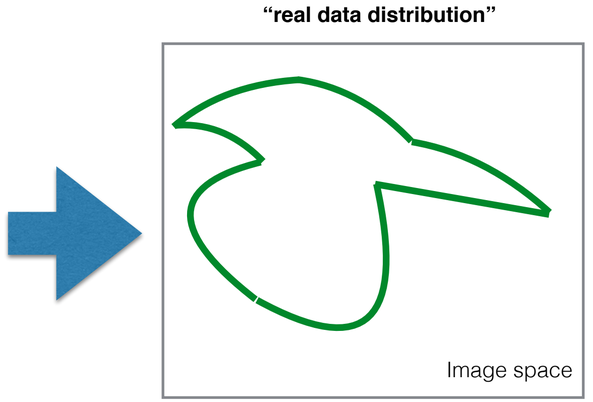
Probability distributions:

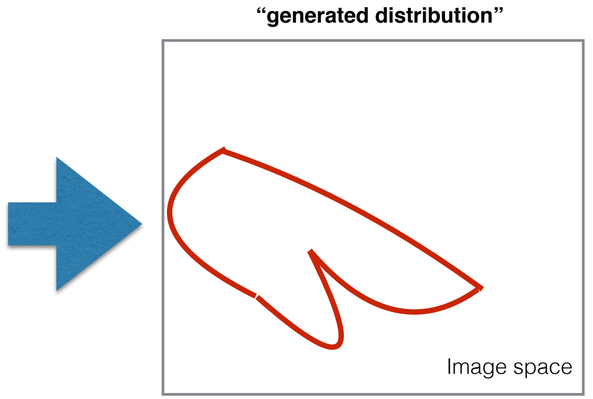
Samples from the “real data distribution”



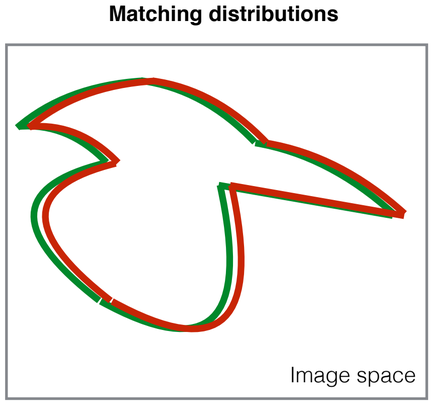
Samples from the “generated distribution”







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

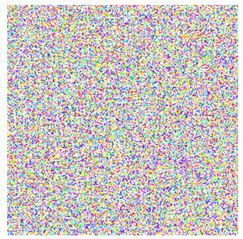
100-d
random code

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

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

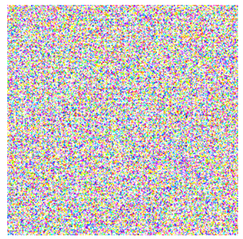
100-d
random code

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

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)**

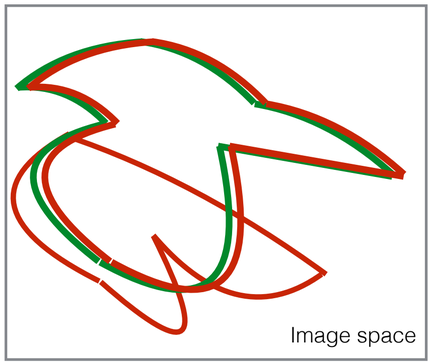
Gradients

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

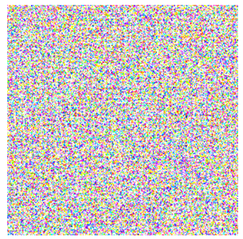
100-d
random code

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

z

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

(64,64,3)
generated image



x

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

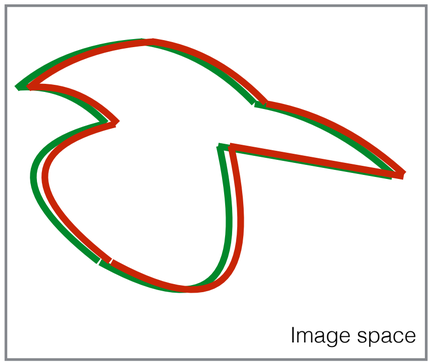
Gradients Binary classification

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

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

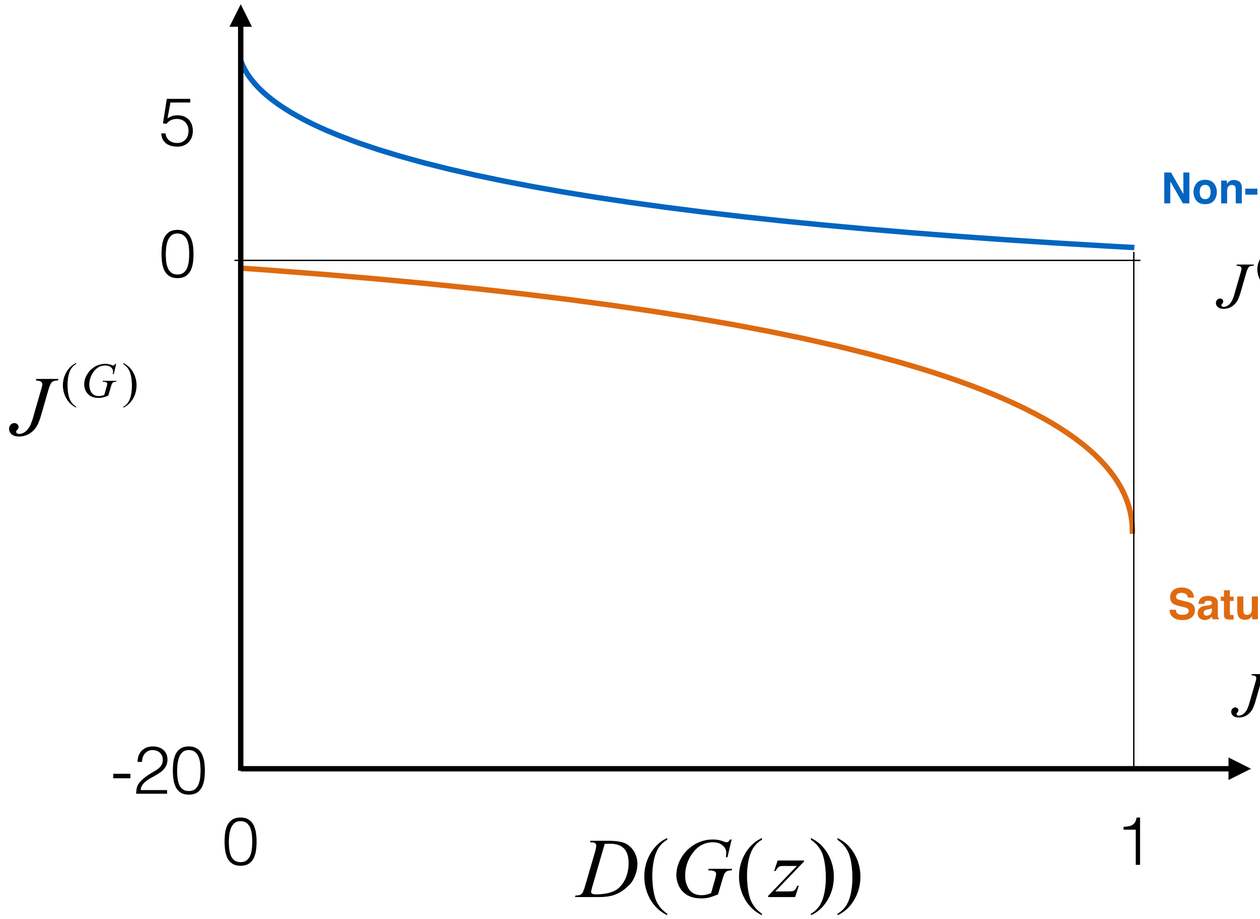
E. In terms of code

[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]

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]

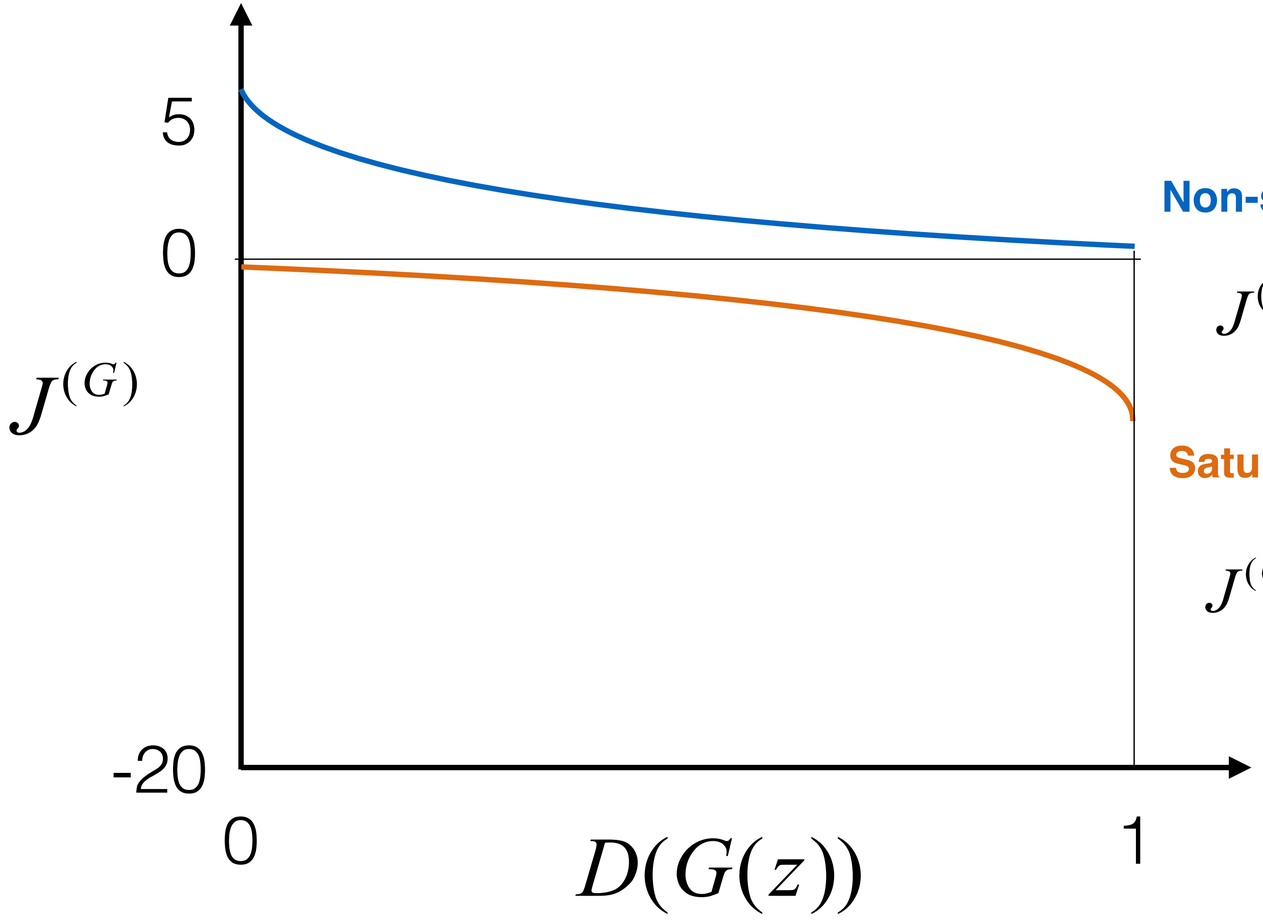
II.C - Training GANs

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

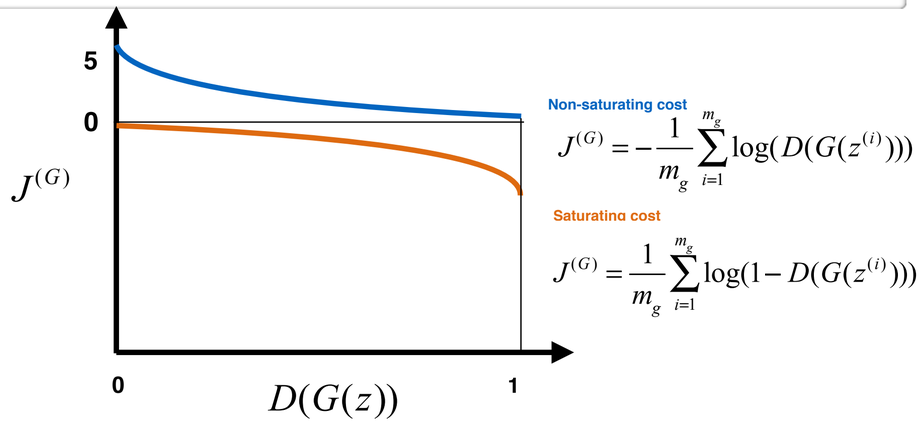
$$J^{(G)} = \frac{1}{m_g}\sum_{i=1}^{m_g}\log(1-D(G(z^{(i)})))$$

[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:

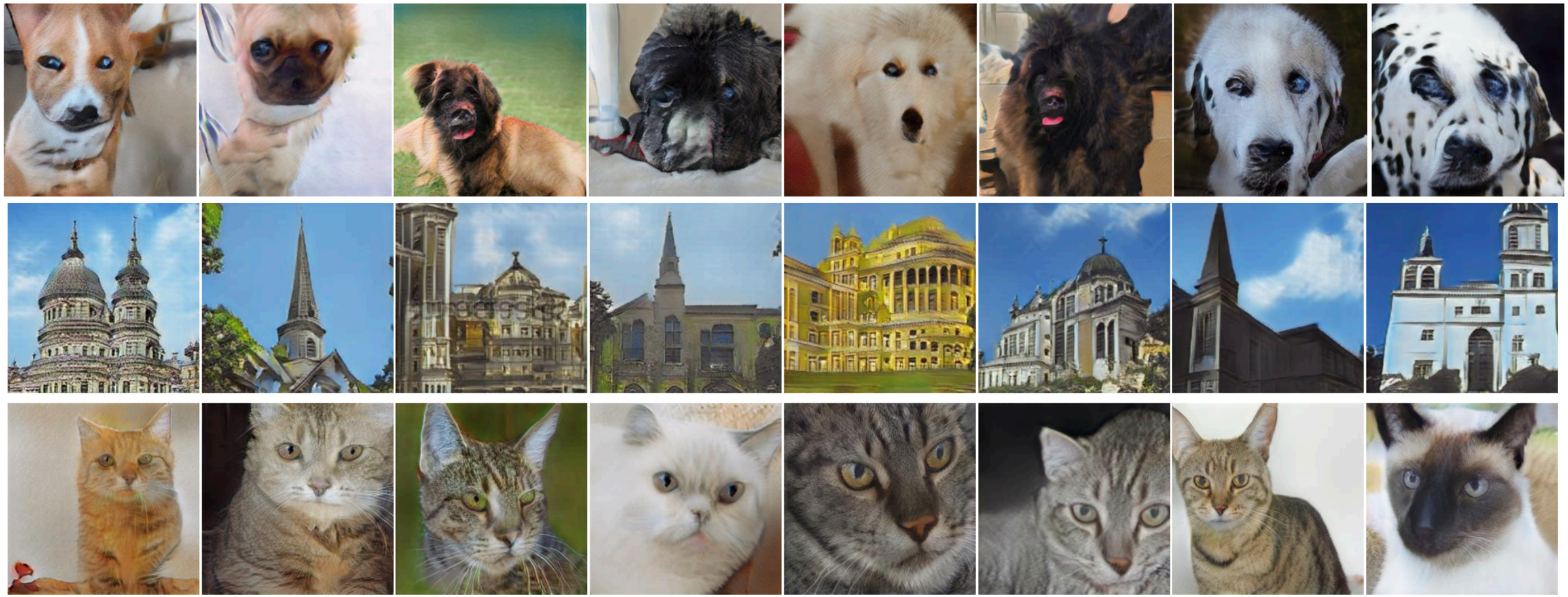
[Karras et al. (2018): A Style-Based Generator Architecture for Generative Adversarial Networks]

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

II.E - Nice results

Image Generation:

Samples from the “generated distribution”



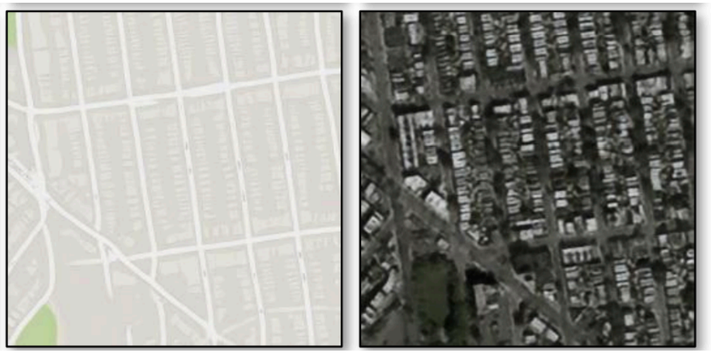
[Zhang et al. (2017): StackGAN++]

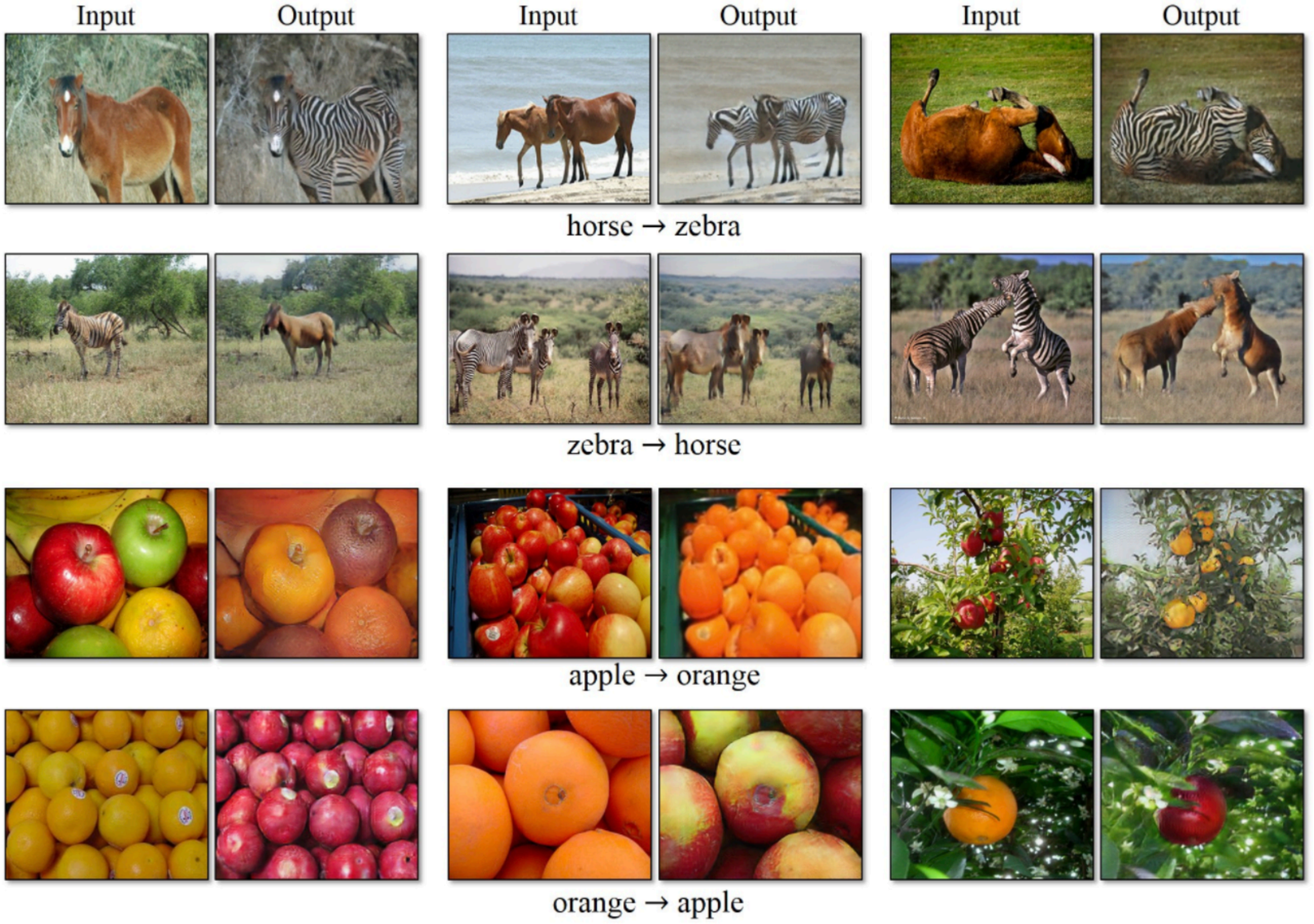
II.E - Nice results

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?**

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?

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

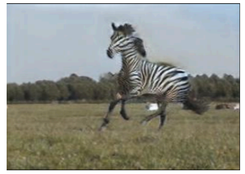
H2Z

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

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))

Discriminator1

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

Generator1 (H2Z)

Z

G2(Z)

Generator2 (Z2H)

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?

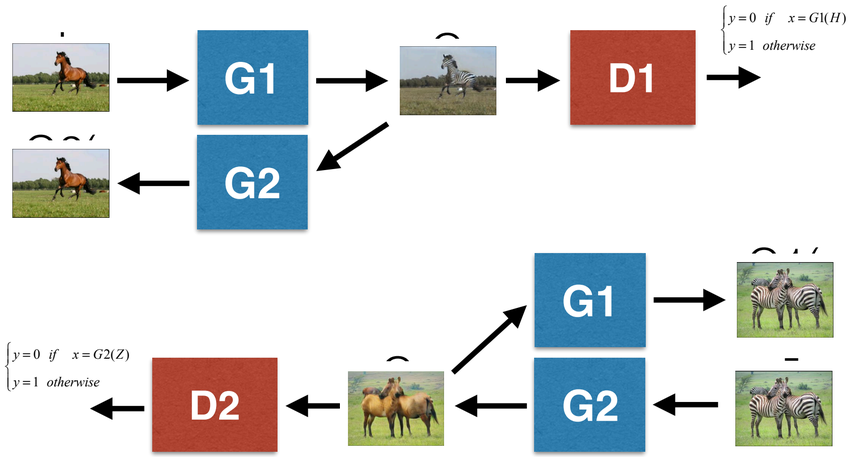
$$J^{(D1)} = -\frac{1}{m_{real}}\sum_{i=1}^{m_{real}}\log(D1(z^{(i)})) - \frac{1}{m_{gen}}\sum_{i=1}^{m_{gen}}\log(1-D1(G1(H^{(i)})))$$

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

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

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

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



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

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 Tuesday 04/29, 9am:

C2M1

- Quiz: Practical aspects of deep learning
- Programming assignment: Initialization
- Programming assignment: Regularization
- Programming assignment: Gradient Checking

C2M2

- Quiz: Optimization Algorithms
- Programming assignment: Optimization

Project Proposal due on Wednesday 04/22 11:59pm PT. You also have until that time to meet with a TA regarding your project.

This Friday 04/24: TA section

Kian Katanforoosh

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]