CS230: Lecture 4
Attacking Networks with Adversarial Examples
-
Generative Adversarial Networks
Kian Katanforoosh
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

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]
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: $\hat{y}(x) = y_{\text{iguana}} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$

2. Defining the loss function

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

3. Optimize the image

After many iterations

$x = x - \alpha \frac{\partial L}{\partial x}$
I. A. Attacking a network with adversarial examples

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

$256^{32 \times 32 \times 3} \approx 10^{7400}$

Space of possible input images

Space of images classified as iguanas

Space of real images

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.

1. Rephrasing what we want:
   
   Find $x$ such that:  
   
   $$\hat{y}(x) = y_{\text{iguana}}$$  
   
   And:  
   
   $$x = x_{\text{cat}}$$

2. Defining the loss function

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

3. Optimize the image

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

[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
I. A. Attacking a network with adversarial examples

$$256^{32 \times 32 \times 3} \approx 10^{7400}$$

- Space of possible input images
- Space of real images
- Space of images that look real to humans
- Space of images classified as iguanas
Adversarial Examples In The Physical World

[Alexey Kurakin, Ian J. Goodfellow, Samy Bengio (2017): Adversarial examples in the physical world]
**I. B. Defenses against adversarial examples**

**Knowledge of the attacker:**

- White-box
- Black-box

**Solution 1**
- Create a SafetyNet

**Solution 2**
- Train on correctly labelled adversarial examples

**Solution 3**
- Adversarial training

\[
x = \text{cat}
\]

\[
y = \text{cat}
\]

\[
L_{\text{new}} = L(W,b,x,y) + \lambda L(W,b,x_{\text{adv}},y)
\]

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

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

Let’s design a method to generate Adversarial Examples

After successful training, we get:

\[ w = (1,3,-1,2,2,3) \]
\[ b = 0 \]

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?

\[ x^* = x + \varepsilon w^T = x + 0.2w^T = (1.2, -0.4, 1.8, 0.4, 3.4, -1.4)^T \]
\[ \hat{y}(x^*) = 0.83 \]
I. C. Why are neural networks vulnerable to adversarial examples?

**Fast Gradient Sign Method:**

\[ x^* = x + \varepsilon \text{sign}(\frac{dJ(W, X, Y)}{dX}) \]
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 $\ll$ amount of data
II.A - Motivation

**Probability distributions:**

Samples from the “real data distribution”

Samples from the “generated distribution”

[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

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

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

\[
\begin{cases}
  y = 0 & \text{if } x = G(z) \\
  y = 1 & \text{otherwise}
\end{cases}
\]

Real images (database) → Generator “G” (Neural Network) → (64,64,3) generated image → Discriminator “D” (Neural Network) → Binary classification

100-d random code

\[
\begin{pmatrix}
  0.47 \\
  \vdots \\
  0.19
\end{pmatrix}
\]
II.B - G/D Game

\[
\begin{align*}
\text{End goal: G is outputting images that are indistinguishable from real images for D}
\end{align*}
\]

\[
\begin{align*}
&\text{100-d random code} \\
&\begin{pmatrix}
0.47 \\
\vdots \\
0.19
\end{pmatrix}
\end{align*}
\]

\begin{align*}
&\text{Generator “G” (Neural Network)} \\
&\text{(64,64,3) generated image}
\end{align*}

\begin{align*}
&\text{Discriminator “D” (Neural Network)} \\
&\begin{cases}
y = 0 & \text{if } x = G(z) \\
y = 1 & \text{otherwise}
\end{cases}
\end{align*}

\begin{align*}
&\text{Gradients} \\
&\text{Probability distribution}
\end{align*}

Real images (database)
II.B - G/D Game

Training procedure, we want to minimize:

- The cost of the discriminator

\[ J^{(D)} = - \frac{1}{m_{\text{real}}} \sum_{i=1}^{m_{\text{real}}} y_{\text{real}}^{(i)} \cdot \log(D(x^{(i)})) - \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} (1 - y_{\text{gen}}^{(i)}) \cdot \log(1 - D(G(z^{(i)}))) \]

  cross-entropy 1:  
  \[ \text{"D should correctly label real data as 1"} \]

  cross-entropy 2:  
  \[ \text{"D should correctly label generated data as 0"} \]

- The cost of the generator

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

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

Labels:
\[ y_{\text{real}} \text{ is always 1} \]
\[ y_{\text{gen}} \text{ is always 0} \]
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:

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

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:

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

New training procedure, we want to minimize:

\[
J^{(D)} = -\frac{1}{m_{\text{real}}} \sum_{i=1}^{m_{\text{real}}} y^{(i)}_{\text{real}} \log(D(x^{(i)})) - \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} (1 - y^{(i)}_{\text{gen}}) \log(1 - D(G(z^{(i)})))
\]

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

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

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

"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.

<table>
<thead>
<tr>
<th>GAN</th>
<th>Discriminator Loss</th>
<th>Generator Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM GAN</td>
<td>$\mathcal{L}<em>D^{\text{GAN}} = - \mathbb{E}</em>{x \sim p_d} [\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$</td>
<td>$\mathcal{L}<em>G^{\text{GAN}} = \mathbb{E}</em>{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$</td>
</tr>
<tr>
<td>NS GAN</td>
<td>$\mathcal{L}<em>D^{\text{NSGAN}} = - \mathbb{E}</em>{x \sim p_d} [\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$</td>
<td>$\mathcal{L}<em>G^{\text{NSGAN}} = - \mathbb{E}</em>{\hat{x} \sim p_g} [\log(D(\hat{x}))]$</td>
</tr>
<tr>
<td>WGAN</td>
<td>$\mathcal{L}<em>D^{\text{WGAN}} = - \mathbb{E}</em>{x \sim p_d} [D(x)] + \mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})]$</td>
<td>$\mathcal{L}<em>G^{\text{WGAN}} = - \mathbb{E}</em>{\hat{x} \sim p_g} [D(\hat{x})]$</td>
</tr>
<tr>
<td>WGAN GP</td>
<td>$\mathcal{L}_D^{\text{WGANGP}} = \mathcal{L}<em>D^{\text{WGAN}} + \lambda \mathbb{E}</em>{\hat{x} \sim p_g} [(</td>
<td></td>
</tr>
<tr>
<td>LS GAN</td>
<td>$\mathcal{L}<em>D^{\text{LSGAN}} = - \mathbb{E}</em>{x \sim p_d} [(D(x) - 1)^2] + \mathbb{E}_{\hat{x} \sim p_g} [D(\hat{x})^2]$</td>
<td>$\mathcal{L}<em>G^{\text{LSGAN}} = - \mathbb{E}</em>{\hat{x} \sim p_g} [(D(\hat{x} - 1)^2]$</td>
</tr>
<tr>
<td>DRAGAN</td>
<td>$\mathcal{L}_D^{\text{DRAGAN}} = \mathcal{L}<em>D^{\text{GAN}} + \lambda \mathbb{E}</em>{\hat{x} \sim p_d + N(0,\sigma)} [(</td>
<td></td>
</tr>
<tr>
<td>BEGAN</td>
<td>$\mathcal{L}<em>D^{\text{BEGAN}} = \mathbb{E}</em>{x \sim p_d} [</td>
<td></td>
</tr>
</tbody>
</table>

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

II.C - Training GANs

for num_iterations:
   for k iterations:
      update D
      update G

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

Non-saturating cost

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

Saturating cost

[Ian Goodfellow (2014): NIPS Tutorial: 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)

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

Non-saturating cost

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

Saturating cost

And a lot more, GANs are hard to train!
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
\[
\begin{bmatrix}
0.12 \\
\vdots \\
0.92
\end{bmatrix}
\]
\(\rightarrow\)
Generator “G” (Neural Network)
\(\rightarrow\)
(64,64,3) generated image

Code 2
\[
\begin{bmatrix}
0.47 \\
\vdots \\
0.19
\end{bmatrix}
\]
\(\rightarrow\)
Generator “G” (Neural Network)
\(\rightarrow\)
(64,64,3) generated image

Code 3
\[
\begin{bmatrix}
0.42 \\
\vdots \\
0.07
\end{bmatrix}
\]
\(\rightarrow\)
Generator “G” (Neural Network)
\(\rightarrow\)
(64,64,3) generated image

\[
\begin{bmatrix}
0.12 \\
\vdots \\
0.92
\end{bmatrix} - \begin{bmatrix}
0.47 \\
\vdots \\
0.19
\end{bmatrix} + \begin{bmatrix}
0.42 \\
\vdots \\
0.07
\end{bmatrix}
\]
\(\rightarrow\)
Generator “G” (Neural Network)

Man with glasses - man + woman = woman with glasses
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]
Goal: Convert horses to zebras on images, and vice-versa.

Data?  Architecture?  Cost?

Unpaired images

Horse images  Zebra images

II.E - Nice results

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

\[
\begin{align*}
H2Z & : \begin{cases} 
 y = 0 & \text{if } x = G2(Z) \\
 y = 1 & \text{otherwise } (x = h)
\end{cases} \\
H & : \text{Generator1 (H2Z)} & \text{Generator2 (Z2H)} & \text{Discriminator1} \\
G2(G1(H)) & & & \text{G1(H)} \\
G2(Z) & & & \text{G1(G2(Z))} \\
G2(Z) & & & \text{Z} \\
G2(G1(H)) & & & \text{Discriminator2} \\
\end{align*}
\]

\[
\begin{align*}
Z2H & : \begin{cases} 
 y = 0 & \text{if } x = G1(H) \\
 y = 1 & \text{otherwise } (x = z)
\end{cases} \\
Z & : \text{Discriminator2} \\
\end{align*}
\]

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

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

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

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

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

\[ J_{\text{cycle}} = \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \| G2(G1(H^{(i)}) - H^{(i)}) \|_1 + \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \| G1(G2(Z^{(i)}) - Z^{(i)}) \|_1 \]

\[ J = J^{(D1)} + J^{(G1)} + J^{(D2)} + J^{(G2)} + \lambda J_{\text{cycle}} \]
CycleGANs:

Face2ramen

+ Face detection

II.E - Nice results

[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

Human Portrait Super Resolution Using GANs

Yujie Shu

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

II.E - Nice results

Motion Retargeting video subjects: https://www.youtube.com/watch?
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…
For Tuesday 10/13, 8.30am:

Complete modules:
- C2M3: Hyperparameter Tuning, Batch Normalization
- C3M1: ML Strategy (1)
- C3M2: ML Strategy (2)

Quizzes:
- Hyperparameter tuning, Batch Normalization, Programming Frameworks
- Bird recognition in the city of Peacetopia (case study)
- Autonomous driving (case study)

Programming Assignments:
- Tensorflow

This Friday 10/9: 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)
sel.combined.compile(loss='binary_crossentropy', optimizer=optimizer)

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]
Kian Katanforoosh
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, img], 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)
II. D. In terms of code

```python
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)
```

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