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

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

2. Defining the loss function

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

3. Optimize the image

   After many iterations

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

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

Kian Katanforoosh
Question: Will the forged image $\mathbf{x}$ 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: $\hat{y}(x) = y_{\text{iguana}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$

And: $x = x_{\text{cat}}$

2. Defining the loss function

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

3. Optimize the image

After many iterations

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

[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 images that look real to humans
- Space of real images
- 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. 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?

\[ x^* = x + \epsilon w = x + 0.2w = (1.2, -0.4, 1.8, 0.4, 3.4, -1.4)^T \]
\[ \hat{y}(x^*) = \sigma(w^T x^*) = \sigma(w^T x + \epsilon | w |^2) = 0.83 \]
I. C. Why are neural networks vulnerable to adversarial examples?

**Fast Gradient Sign Method:**

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

W: Whitebox  
B: Blackbox  
T: Targeted  
NT: Non-targeted
I. B. Defenses against adversarial examples

Knowledge of the attacker:

- White-box
- Black-box

Examples of defenses (exploratory)

- Create a SafetyNet
- Train on correctly labelled adversarial examples
- Adversarial training

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

R: Robustness  
D: Detection
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”

Matching distributions

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

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

[Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaoolei 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)

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

\[x = \begin{cases} 
0 & \text{if } x = G(z) \\
1 & \text{otherwise} 
\end{cases}\]
II.B - G/D Game

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

\[
\begin{align*}
&\begin{pmatrix}
0.47 \\
\vdots \\
0.19 \\
\end{pmatrix} \\
\Rightarrow \\
&\text{Generator “G” (Neural Network)} \\
\Rightarrow \\
&\text{(64,64,3) generated image} \\
\Rightarrow \\
\begin{align*}
&x \\
\Rightarrow \\
&\text{Discriminator “D” (Neural Network)} \\
\Rightarrow \\
&\begin{cases}
y = 0 & \text{if } x = G(z) \\
y = 1 & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{pmatrix}
\text{Real images (database)} \\
\Rightarrow \\
\text{Image space} \\
\Rightarrow \\
\text{Probability distribution}
\end{pmatrix}
\]
II.B - G/D Game

Training procedure, we want to minimize:

The cost of the discriminator

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

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

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

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: \[ \begin{cases} y_{\text{real}} \text{ is always 1} \\ y_{\text{gen}} \text{ is always 0} \end{cases} \]
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:

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

Non-saturating cost

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

Saturating cost

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

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

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

New training procedure, we want to minimize:

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

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

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

“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>$\mathcal{L}<em>D^{\text{MMGAN}} = -\mathbb{E}</em>{x \sim p_d} [\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$</td>
<td>$\mathcal{L}<em>G^{\text{MMGAN}} = \mathbb{E}</em>{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$</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>$\mathcal{L}_D^{\text{WANGP}} = \mathcal{L}<em>D^{\text{WGAN}} + \lambda \mathbb{E}</em>{\hat{x} \sim p_g} [(|\nabla D(\alpha x + (1 - \alpha)\hat{x})|_2 - 1)^2]$</td>
<td>$\mathcal{L}<em>G^{\text{WANGP}} = -\mathbb{E}</em>{\hat{x} \sim p_g} [D(\hat{x})]$</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>$\mathcal{L}_D^{\text{DRAGAN}} = \mathcal{L}<em>D^{\text{GAN}} + \lambda \mathbb{E}</em>{\hat{x} \sim p_d + N(0,\sigma)} [(|\nabla D(\hat{x})|_2 - 1)^2]$</td>
<td>$\mathcal{L}<em>G^{\text{DRAGAN}} = \mathbb{E}</em>{\hat{x} \sim p_g} [\log(1 - D(\hat{x}))]$</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
II.C - Training GANs

Simultaneously training G/D?

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

Non-saturating cost

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

Saturating cost

for num_iterations:
  for k iterations:
    update D
    update G

[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)

And a lot more, GANs are hard to train!

II.C - Training GANs

Non-saturating cost

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

Saturating cost

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

[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
\[
\begin{pmatrix}
0.12 \\
\vdots \\
0.92
\end{pmatrix}
\]

\rightarrow

Generator “G” (Neural Network)

\rightarrow

(64,64,3) generated image

Code 2
\[
\begin{pmatrix}
0.47 \\
\vdots \\
0.19
\end{pmatrix}
\]

\rightarrow

Generator “G” (Neural Network)

\rightarrow

(64,64,3) generated image

Code 3
\[
\begin{pmatrix}
0.42 \\
\vdots \\
0.07
\end{pmatrix}
\]

\rightarrow

Generator “G” (Neural Network)

\rightarrow

(64,64,3) generated image

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

\rightarrow

Generator “G” (Neural Network)

\rightarrow

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.
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.

<table>
<thead>
<tr>
<th>Data?</th>
<th>Architecture?</th>
<th>Cost function?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpaired images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Horse images</td>
<td>Zebra images</td>
<td></td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
  &\text{H2Z} \\
  &\begin{cases}
    y = 0 & \text{if } x = G2(Z) \\
    y = 1 & \text{otherwise } (x = h)
  \end{cases}
\end{align*}
\]

\[
\begin{align*}
  &\text{Generator1 (H2Z)} \\
  &\text{Discriminator1}
\end{align*}
\]

\[
\begin{align*}
  &\text{Generator2 (Z2H)} \\
  &\text{Discriminator2}
\end{align*}
\]

\[
\begin{align*}
  &\text{Discriminator2}
\end{align*}
\]

\[
\begin{align*}
  &\text{G2(Z)} \\
  &\begin{cases}
    y = 0 & \text{if } x = G2(Z) \\
    y = 1 & \text{otherwise } (x = h)
  \end{cases}
\end{align*}
\]

\[
\begin{align*}
  &\text{II.E - Nice results}
\end{align*}
\]
Loss to minimize?

\[ J^{(D_1)} = -\frac{1}{m_{\text{real}}} \sum_{i=1}^{m_{\text{real}}} \log(D_1(z^{(i)})) - \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \log(1 - D_1(G_1(H^{(i)}))) \]

\[ J^{(G_1)} = -\frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \log(D_1(G_1(H^{(i)}))) \]

\[ J^{(D_2)} = -\frac{1}{m_{\text{real}}} \sum_{i=1}^{m_{\text{real}}} \log(D_2(h^{(i)})) - \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \log(1 - D_2(G_2(Z^{(i)}))) \]

\[ J^{(G_2)} = -\frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \log(D_2(G_2(Z^{(i)}))) \]

\[ J_{\text{cycle}} = \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \| G_2(G_1(H^{(i)}) - H^{(i)}) \|_1 + \frac{1}{m_{\text{gen}}} \sum_{i=1}^{m_{\text{gen}}} \| G_1(G_2(Z^{(i)}) - Z^{(i)}) \|_1 \]

\[ J = J^{(D_1)} + J^{(G_1)} + J^{(D_2)} + J^{(G_2)} + \lambda J_{\text{cycle}} \]
II.E - Nice results

CycleGANs:

Face2ramen

[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://affinelaye.com/pixsrv/ by Christopher Hesse.
II.E - Nice results

Human Portrait Super Resolution Using GANs

Yujie Shu

Figure 1: Input LR 32x32, SRPRGAN 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…
Announcements

For Wednesday 10/20, 9/30am PDT:

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

Quizzes (due at 9 30 am PST (right before lecture)):
- Hyperparameter tuning, Batch Normalization, Programming Frameworks
- Bird recognition in the city of Peacetopia (case study)
- Autonomous driving (case study)

Programming Assignments (due at 9 30 am PST (right before lecture)):
- Tensorflow

This Friday 10/15: 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)

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

```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)

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)
```

[Kian Katanforoosh](https://github.com/eriklindernoren/Keras-GAN)
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))
    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))
    g_loss = self.combined.train_on_batch(noise, valid)
```

---

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