Advanced Section #8:
Generative Adversarial Networks (GANs)

CS109B Data Science 2
Vincent Casser  Pavlos Protopapas
Outline

- Concept and Math
- Applications
- Common Problems
- Wasserstein GANs, Conditional GANs and CycleGANs
- Troubleshooting GANs
- Hands-on: Building an Image GAN in Keras
- Influential Papers and References
**Concept**

**Generator**

Job: Fool discriminator

```
“Both are pandas!”
```

- Real
- Generated

**Discriminator**

Job: Catch lies of the generator

```
“Nope”
```

- Confidence: 0.9997
- Confidence: 0.1617
Concept

Generator

Job: Fool discriminator

Generated

Real

“Both are pandas!”

Discriminator

Job: Catch lies of the generator

Confidence: 0.3759

Confidence: 1.0

“Good try...”
GAN Structure

**Generator**

- Job: Fool discriminator
- Noise $z$ → Sample $G(z)$

**Discriminator**

- Job: Catch lies of the generator
- Sample $x$ (real) → Score $D(x) \rightarrow 1$
- Sample $G(z)$ (fake) → Score $D(G(z)) \rightarrow 0$
Math in a nutshell

**Generator**

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

- \(m\): Number of samples
- \(z\): Random noise samples
- \(x\): Real samples

How realistic are the generated samples? G wants to maximize this.

**Discriminator**

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

- Make sure real samples are classified as being real.
- Make sure generated samples are classified as unreal.

D wants to maximize this. D wants to minimize this.
Math in a nutshell

Generator

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

m: Number of samples
z: Random noise samples
x: Real samples

Discriminator

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

\[
\begin{array}{|c|c|}
\hline
\text{Generator} & - & 1.0 \\
\hline
\text{Discriminator} & 1.0 & 0.0 \\
\hline
\end{array}
\]

Targets

D(x) = 0.9997
D(G(z)) = 0.1617
Applications

● (Conditional) synthesis
  ○ Font generation
  ○ Text2Image
  ○ 3D Object generation

● Data augmentation
  ○ Aiming to reduce need for labeled data
  ○ GAN is only used as a tool enhancing the training process of another model

● Style transfer and manipulation
  ○ Face Aging
  ○ Painting
  ○ Pose estimation and manipulation
  ○ Inpainting
  ○ Blending

● Signal super resolution
### Applications: Style Transfer and Manipulation

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Input Image" /></td>
<td><img src="image2.png" alt="Output Image" /></td>
<td><img src="image3.png" alt="Input Image" /></td>
<td><img src="image4.png" alt="Output Image" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="Input Image" /></td>
<td><img src="image6.png" alt="Output Image" /></td>
<td><img src="image7.png" alt="Input Image" /></td>
<td><img src="image8.png" alt="Output Image" /></td>
</tr>
<tr>
<td><img src="image9.png" alt="Input Image" /></td>
<td><img src="image10.png" alt="Output Image" /></td>
<td><img src="image11.png" alt="Input Image" /></td>
<td><img src="image12.png" alt="Output Image" /></td>
</tr>
<tr>
<td><img src="image13.png" alt="Input Image" /></td>
<td><img src="image14.png" alt="Output Image" /></td>
<td><img src="image15.png" alt="Input Image" /></td>
<td><img src="image16.png" alt="Output Image" /></td>
</tr>
</tbody>
</table>
Applications: Style Transfer and Manipulation

Input | Output
--- | ---
Horse | Zebra
Zebra | Horse
Apple | Orange
Orange | Apple
Applications: Style Transfer and Manipulation

winter Yosemite → summer Yosemite

summer Yosemite → winter Yosemite
Applications: Style Transfer and Manipulation

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Input Image" /></td>
<td><img src="image2" alt="Output Image" /></td>
<td><img src="image3" alt="Input Image" /></td>
<td><img src="image4" alt="Output Image" /></td>
<td><img src="image5" alt="Input Image" /></td>
<td><img src="image6" alt="Output Image" /></td>
</tr>
<tr>
<td><img src="image7" alt="Input Image" /></td>
<td><img src="image8" alt="Output Image" /></td>
<td><img src="image9" alt="Input Image" /></td>
<td><img src="image10" alt="Output Image" /></td>
<td><img src="image11" alt="Input Image" /></td>
<td><img src="image12" alt="Output Image" /></td>
</tr>
<tr>
<td><img src="image13" alt="Input Image" /></td>
<td><img src="image14" alt="Output Image" /></td>
<td><img src="image15" alt="Input Image" /></td>
<td><img src="image16" alt="Output Image" /></td>
<td><img src="image17" alt="Input Image" /></td>
<td><img src="image18" alt="Output Image" /></td>
</tr>
</tbody>
</table>
Applications: Style Transfer and Manipulation
Applications: Style Transfer and Manipulation
Applications: Style Transfer and Manipulation
Applications: Style Transfer and Manipulation
Applications: Style Transfer and Manipulation
Applications: Style Transfer and Manipulation
Applications: Signal Super Resolution

LG Image                       Generated Image
Applications: Signal Super Resolution

43074 from BSD100
(PSNR / Perceptual Index)

HR
(∞ / 2.31)

Bicubic
(29.29 / 7.35)

SRCNN
(29.62 / 6.46)

EDSR
(29.76 / 6.25)

RCAN
(29.79 / 6.22)

EnhanceNet
(27.69 / 3.00)

SRGAN
(27.29 / 2.74)

ESRGAN
(27.69 / 2.76)
Common Problems: Oscillation

- Both generator and discriminator jointly searching for equilibrium, but model updates are independent.
- No theoretical convergence guarantees.
- Solution: Extensive hyperparameter-search, sometimes manual intervention.
Common Problems: Vanishing gradient

- Discriminator can become too strong to provide signal for the generator.
- Generator can learn to fool the discriminator consistently.
- Solution: Do (not) pretrain discriminator, or lower its learning rate relatively to the generator. Change the number of updates for generator/discriminator per iteration.
GANs and Game Theory

- Original GAN formulation based on zero-sum non-cooperative game.
- If one wins, the other one loses (minimax).
- GANs converge when G and D reach a Nash equilibrium: the optimal point of

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_x(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]
Common Problems: Mode collapse

- Generator can collapse so that it always produces the same samples.
- Generator restrained to small subspace generating samples of low diversity.
- Solution: Encourage diversity through minibatch discrimination (presenting the whole batch to the discriminator for review) or feature matching (add generator penalty for low diversity), or use multiple GANs
Common Problems: Evaluation metrics

- GANs are still evaluated on a very qualitative basis.
- Defining proper metrics is challenging. How does a “good” generator look like?
- Solution: Active research field and domain specific. Strong classification models are commonly used to judge the quality of generated samples

Inception score
TSTR score
Wasserstein GAN

- Using the standard GAN formulation, training is extremely unstable.
- Discriminator often improves too quickly for the generator to catch up.
- Careful balancing is needed.
- Mode collapse is frequent.

**WGAN (Wasserstein GAN):**
Arjovsky, M., Chintala, S. and Bottou, L., 2017. Wasserstein GAN.
Distance is everything:
In general, generative models seek to minimize the distance between real and learned distribution.

Wasserstein (also EM, Earth-Mover) distance:

“Informally, if the distributions are interpreted as two different ways of piling up a certain amount of dirt over the region D, the Wasserstein distance is the minimum cost of turning one pile into the other; where the cost is assumed to be amount of dirt moved times the distance by which it is moved.”
Wasserstein GAN

- Exact computation is intractable.
- Idea: Use a CNN to **approximate** Wasserstein distance.
- Here, we re-use the discriminator, whose outputs are now **unbounded**
- We define a custom loss function, in Keras:

\[ \text{K.mean}(y\_\text{true} \times y\_\text{pred}) \]

\( y\_\text{true} \) here is chosen from \{-1, 1\} according to real/fake

Idea: make predictions for one type as large as possible, for others as small as possible
Wasserstein GAN

The authors claim:

- Higher stability during training, less need for carefully balancing generator and discriminator.
- Meaningful loss metric, correlating well with sample quality.
- Mode collapse is rare.
Wasserstein GAN

Tips for implementing Wasserstein GAN in Keras.

- Leave the discriminator output unbounded, i.e. apply linear activation.
- Initialize with small weights to not run into clipping issues from the start.
- Remember to run sufficient discriminator updates. This is crucial in the WGAN setup.
- You can use the wasserstein surrogate loss implementation below.
- Clip discriminator weights by implementing your own keras constraint.

```python
def wasserstein_loss(y_true, y_pred):
    return K.mean(y_true * y_pred)
```

```python
class WeightClip(keras.constraints.Constraint):
    def __init__(self, c):
        self.c = c

    def __call__(self, p):
        return K.clip(p, -self.c, self.c)

    def get_config(self):
        return {'name': self.__class__.__name__, 'c': self.c}
```
CycleGAN

$G_{AB}$: Generates a fake image in domain B from a real image in domain A.

$G_{BA}$: Generates a reconstructed image of domain A from a horse image generated from a zebra.

This makes the shape to be maintained when $G_{AB}$ generates a horse image from the zebra.
CycleGAN

Generator $G_{AB}$ learns to sneak in information for $G_{BA}$.
As in VAEs, GANs can simply be conditioned to generate a certain mode of data.
Troubleshooting GANs

GANs can be frustrating to work with. Here are some tips for your reference:

- **Models.** Make sure models are correctly defined. You can debug the discriminator alone by training on a vanilla image-classification task.
- **Data.** Normalize inputs properly to [-1, 1]. Make sure to use tanh as final activation for the generator in this case.
- **Noise.** Try sampling the noise vector from a normal distribution (not uniform).
- **Normalization.** Apply BatchNorm when possible, and send the real and fake samples in separate mini-batches.
- **Activations.** Use LeakyRelu instead of Relu.
- **Smoothing.** Apply label smoothing to avoid overconfidence when updating the discriminator, i.e. set targets for real images to less than 1.
- **Diagnostics.** Monitor the magnitude of gradients constantly.
- **Vanishing gradients.** If the discriminator becomes too strong (discriminator loss = 0), try decreasing its learning rate or update the generator more often.
Building an Image GAN

- Training a GAN can be frustrating and time-intensive.
- We will walk through a clean minimal example in Keras.
- Results are only on proof-of-concept level to enhance understanding. For state-of-the-art GANs, see references.

In the code example, if you don’t tune parameters carefully, you won’t surpass this level by much:
Building an Image GAN: Discriminator

Takes an image \([H, W, C]\) and outputs a vector of \([M]\), either class scores (classification) or single score quantifying photorealism. Can be any image classification network, e.g. ResNet or DenseNet. We use a minimalistic custom architecture.
Building an Image GAN: Generator

Takes a vector of noise $[N]$ and outputs an image of $[H, W, C]$. Network has to perform synthesis. Again, we use a very minimalistic custom architecture.

*Source*

**Noise vector**

In practice, the projection is usually done using a dense of $H \times W \times C$ units, followed by a reshape operation. You might want to regularize this part well.
Building an Image GAN: Full Setup

It is important to define the models properly in Keras, so that the weights of the respective models are fixed at the right time.

1. Define the discriminator model, and compile it.
2. Define the generator model, no need to compile.
3. Define an overall model comprised of these two, setting the discriminator to not trainable before the compilation:

```python
model = keras.Sequential()
model.add(generator)
model.add(discriminator)
discriminator.trainable = False
model.compile(…)
```
Building an Image GAN: Training Loop

The training loop has to be executed manually:

1. **Select** R real images from the training set.
2. **Generate** F fake images by sampling random vectors of size N, and predicting images from them using the generator.
3. **Train the discriminator** using \texttt{train\_on\_batch}: call it separately for the batch of R real images and F fake images, with the groundtruth being 1 and 0, respectively.
4. **Sample** new random vectors of size N.
5. **Train the full model** on the new vectors using \texttt{train\_on\_batch} with targets of 1. This will update the generator.
Building an Image GAN: Training Progress
<table>
<thead>
<tr>
<th>Influential GAN-Papers (in order)</th>
</tr>
</thead>
</table>
## Influential GAN-Papers (in order)

<table>
<thead>
<tr>
<th>Paper Title</th>
<th>Year</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCGAN</td>
<td>SGAN</td>
<td>AffGAN</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>WGAN</td>
<td>SimGAN</td>
<td>TP-GAN</td>
</tr>
<tr>
<td>CGAN</td>
<td>VGAN</td>
<td>IcGAN</td>
</tr>
<tr>
<td>LAPGAN</td>
<td>iGAN</td>
<td>ID-CGAN</td>
</tr>
<tr>
<td>SRGAN</td>
<td>3D-GAN</td>
<td>AnoGAN</td>
</tr>
<tr>
<td>CycleGAN</td>
<td>CoGAN</td>
<td>LS-GAN</td>
</tr>
<tr>
<td>WGAN-GP</td>
<td>CatGAN</td>
<td>Triple-GAN</td>
</tr>
<tr>
<td>EBGAN</td>
<td>MGAN</td>
<td>TGAN</td>
</tr>
<tr>
<td>VAE-GAN</td>
<td>S^2GAN</td>
<td>BS-GAN</td>
</tr>
<tr>
<td>BiGAN</td>
<td>LSGAN</td>
<td>MalGAN</td>
</tr>
</tbody>
</table>
Sources and References

Run BigGAN in COLAB:
 gan_generation_with_tf_hub.ipynb

k/
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
https://github.com/tensorlayer/srgan
https://junyanz.github.io/CycleGAN/
https://affinelayer.com/pixsrv/
https://tcwang0509.github.io/pix2pixHD/