Generative Learning: Generative Adversarial Networks (GANs)

Yossi Keshet
Bar-Ilan University
May 11, 2020
Motivation

PixelRNN and PixelCNN define *tractable* density function, and optimize likelihood of training data:

\[ p(x) = \prod_{i=1}^{N} p(x_i|x_{i-1}, x_{i-2}, \ldots, 1) \]

Variational autoencoders (VAEs) define *intractable* density function with latent variable \( z \):

\[ p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \]

Can we skip the explicit density estimation and just generate samples?
Generative Adversarial Networks (GAN)

**Problem:** There is no direct and effective way to sample from unknown complex high-dimensional distribution.

**Solution:** We will sample from a *simple* distribution (e.g., isotropic Normal distribution), and *learn the transformation* to the training distribution.

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
Generative Adversarial Networks (GAN)

Problem: There is no direct and effective way to sample from unknown complex high-dimensional distribution.

Solution: We will sample from a simple distribution (e.g., isotropic Normal distribution), and learn the transformation to the training distribution.
Generative Adversarial Networks (GAN)

Problem: There is no direct and effective way to sample from unknown complex high-dimensional distribution.

Solution: We will sample from a simple distribution (e.g., isotropic Normal distribution), and learn the transformation to the training distribution.

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
GAN: two-player game

**Generator network**: try to generating real-looking images and fool the discriminator

**Discriminator network**: try to distinguish between real and fake images
GAN: two-player game

**Generator network**: try to generating real-looking images and fool the discriminator

**Discriminator network**: try to distinguish between real and fake images
GAN: two-player game

**Generator network:** try to generating real-looking images and fool the discriminator

**Discriminator network:** try to distinguish between real and fake images

Train jointly the generator network $G_{\theta_G}$ and the discriminator network $D_{\theta_D}$:

$$\min_{\theta_G} \max_{\theta_D} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_D}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right]$$

where:

- The likelihood of the discriminator $D$ on real data should be high:
  $$\max_{\theta_D} \mathbb{E}_{x \sim p_{data}} \log D_{\theta_D}(x)$$

- The likelihood of the discriminator on the fake data $\tilde{x} = G_{\theta_G}(z)$ should be low:
  $$\max_{\theta_D} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z)))$$

- The data generated by the generator $G$ should fool the discriminator:
  $$\min_{\theta_G} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z)))$$
GAN: two-player game

The objective:

$$\min_{\theta_G} \max_{\theta_D} \left[ \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_D}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right]$$

is solved by alternating between

1. Gradient ascent on $\theta_D$

$$\max_{\theta_D} \left[ \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_D}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right]$$

2. Gradient decent on $\theta_G$

$$\min_{\theta_G} \left[ \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right]$$
GAN: two-player game

In practice, optimizing this generator objective does not work well.

For $\log(1 - D_{\theta_D}(G_{\theta_G}(z)))$ the gradient is very flat when the image is likely to be fake, and steep when the sample is already good.

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good
GAN: two-player game

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Instead of

\[ \min_{\theta_G} \left[ \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right] \]

we maximize

\[ \max_{\theta_G} \left[ \mathbb{E}_{z \sim p(z)} \log(D_{\theta_D}(G_{\theta_G}(z))) \right] \]
GAN: two-player game

$$\min_{\theta_G} \left[ \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right] \rightarrow \max_{\theta_G} \left[ \mathbb{E}_{z \sim p(z)} \log(D_{\theta_D}(G_{\theta_G}(z))) \right]$$
for number of training iterations do
    for k steps do
        • Sample minibatch of \( m \) noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
        • Sample minibatch of \( m \) examples \( \{x^{(1)}, \ldots, x^{(m)}\} \) from data generating distribution \( p_{\text{data}}(x) \).
        • Update the discriminator by ascending its stochastic gradient:
          \[
          \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D \left( x^{(i)} \right) + \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right) \right].
          \]
    end for
    • Sample minibatch of \( m \) noise samples \( \{z^{(1)}, \ldots, z^{(m)}\} \) from noise prior \( p_g(z) \).
    • Update the generator by descending its stochastic gradient:
      \[
      \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right).
      \]
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
After training we use only the generator to generate new images based on a random sample of $z$. 

![Diagram of GAN inference]

- **Real or Fake**
  - Discriminator Network
  - Fake Images (from generator)
  - Real Images (from training set)
- **Random noise**
  - $z$
GAN example 1

MNIST generated examples. Rightmost column shows the nearest training example of the neighboring sample.
GAN example 2

Faces generation (data: TFD)
GAN example 3

CIFAR-10 (fully connected model)
GAN example 4

CIFAR-10 (convolutional discriminator and “deconvolutional” generator)
GAN with convolutional architectures (DCGAN)

Generator $G$ is upsampling network with \textit{fractionally-strided} convolutions.

Discriminator $D$ is a convolutional neural network.

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

GAN with convolutional architectures (DCGAN)

Generator $G$ is upsampling network with fractionally-strided convolutions.

Discriminator $D$ is a convolutional neural network.

DCGAN examples

Generated bedrooms after one training pass through the dataset
DCGAN examples

Generated bedrooms after 5 epochs of training
DCGAN examples

Interpolation between a series of 9 random points in $Z$
DCGAN examples

Smiling woman  Neutral woman  Neutral man

Samples from the model
DCGAN examples

- Smiling woman
- Neutral woman
- Neutral man

Samples from the model

Average Z vectors, do arithmetic
DCGAN examples

Samples from the model
Average Z vectors, do arithmetic

Smiling woman  Neutral woman  Neutral man  =  Smiling Man
DCGAN examples

man with glasses  +  woman without glasses = woman with glasses

man without glasses