Generative Adversarial Networks

Some slides from Fei-Fei Li, Justin Johnson, Serena Yeung, and Roger Grosse
Reminder: Autoencoders

- Learn to compute a code that can be used to generate a reconstruction
- The reconstructions are generally blurry
- Try to minimize the squared difference between the input and the reconstruction
How does the decoder work?

- $W^{(7,i,:)}$ is the i-th template for the image
- The second-to-last layer defines the coefficients for each of the templates
- The code contains information about how to compute those coefficients
  - (For faces) Whose face is it?
  - (For faces) Which way is the person looking?
Example generated images:

• Generated using a variant of autoencoders
  https://www.youtube.com/watch?v=XNZIN7Jh3Sg
Why are the outputs blurry for vanilla autoencoders?

• 700 global templates isn’t bad if we want to reconstruct faces 64x64 in size
  • Don’t even need a deep architecture

• 700 global templates is pretty bad if we want to reconstruct large images
  • Want to get the details in the image right
Local & Hierarchical Templates

• Want to start with the code and build up the output images

• Want to build up the image from local templates
  • Stitch the images together from plausible image patches instead of averaging global templates
  • Want to output an image of an eye at potentially a lot of locations, just store information about what eyes look like once.

→ Convolutions.
A generator with fractionally-strided convolutions
Partially-strided convolutions

- Can get a $4 \times 4$ output from a $2 \times 2$ input by zero padding
- A more efficient way of accomplishing the same thing:
  - If a convolution can be computed using $A = CB$ where $C$ is a large matrix (the weights of the convolution kernel arranged so that things work out), we can compute $C^T A$ to get a matrix that has the same size as $B$
A probabilistic generative model

• To generate a random image
  • Sample \( z \sim N(0, I) \)
    • Each coordinate in \( z \) determines the content of the image
  • Run the \( z \) though the decoder
Training deep autoencoders

• Training deep autoencoders is difficult and doesn’t work very well

• Convolutions and down-sampling means exact location information is lost

• An active research area
Generative Adversarial Nets (GANs)

• Idea: train two networks
  • **Generator network**: try to fool the discriminator by generating real-looking images
  • **Discriminator network**: try to distinguish between real and fake images
Training GANs: Two-Player Game

• Play a minimax game: given that the discriminator will try to do the best job it can, the generator is set to make the discriminator as wrong as possible.

• The discriminator outputs a probability
Training GANs: Two-Player Game

\[
\min_{\theta_g} \max_{\theta_d} \left[ E_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]
\]

- \(x\) is randomly sampled from the training data. The discriminator wants to output 1.
- \(z\) is randomly sampled, and then a fake image is generated by the generator from the code \(z\). The discriminator wants to output 0.
Training GANs: a Two-player game

$$\min_{\theta_g} \max_{\theta_d}[E_{x \sim p_{data}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log \left( 1 - D_{\theta_d} \left( G_{\theta_g}(z) \right) \right)]$$

- Alternate between:
  1. **Gradient ascent** for the discriminator
     $$\max_{\theta_d}[E_{x \sim p_{data}} \log D_{\theta_d}(x) + E_{z \sim p(z)} \log \left( 1 - D_{\theta_d} \left( G_{\theta_g}(z) \right) \right)]$$
     Do a better job outputting 1 on real images and 0 on fake images
  2. **Gradient descent** on the generator
     $$\min_{\theta_g} E_{z \sim p(z)} \log \left( 1 - D_{\theta_d} \left( G_{\theta_g}(z) \right) \right)$$
     Do a better job making sure the discriminator outputs large numbers on fake images
Modifying the cost function

\[
\min_{\theta_g} E_{z \sim p(z)} \log \left( 1 - D_{\theta_d} \left( G_{\theta_g}(z) \right) \right)
\]

• Problem: if the generator is doing a bad job and the discriminator knows it, it’s hard to learn from that
  • Modified cost:
    \[
    J = E_{z \sim p(z)} - \log \left( D_{\theta_d} \left( G_{\theta_g}(z) \right) \right)
    \]
  • Now, the discriminator is doing poorly for code \( z \), \( \frac{\partial J}{\partial D_{\theta_d}(G_{\theta_g}(z))} \) is large, so that the update to \( \theta_d \) is large
Training in practice

• Sample real images from the train set to estimate
  \[ E_{x \sim p_{data}} \log D_{\theta_d}(x) \approx \frac{1}{n} \sum_i \log D_{\theta_d}(x^{(i)}) \]

• Sample fake images (by first sampling code \( z \) and then generating images) to estimate
  \[ E_{z \sim p(z)} - \log \left( D_{\theta_d} \left( G_{\theta_g}(z) \right) \right) \approx -\frac{1}{n} \sum_j \log \left( D_{\theta_d} \left( G_{\theta_g}(z^{(j)}) \right) \right) \]

• Can compute the gradients now!
Training GANs

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 $p_{data}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
      $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by ascending its stochastic gradient (improved objective):
    $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$
end for
Training a GAN

Real samples …… are far away from generated sample distribution —
The discriminator probability … is high for real samples and low for fake samples
Training a GAN

Real samples …… are far away from generated sample distribution –
The discriminator probability … is high for real samples and low for fake samples
Training a GAN

Real samples ⋅⋅⋅⋅ are close to the generated sample distribution –
The discriminator probability ⋅⋅ is high for most real samples but not all and high for some fake samples
Training a GAN

Real samples …… are very close to the generated sample distribution –
The discriminator probability … is constant since the discriminator can’t tell real from fake samples
Initial results: Generated Images

Nearest examples from train set

Goodfellow et al. 2014
Convolutional Architecture: Generated images

Radford et al. 2016
Interpolating between random points in latent (\(z\)) space

\[ z = z_0 \quad \quad z = 0.2z_0 + 0.8z_1 \quad \quad z = z_1 \]

Radford et al. 2016
Vector Arithmetic in $z$ space

Radford et al, ICLR 2016

Samples from the model

Average $Z$ vectors, do arithmetic

Smiling woman

Neutral woman

Neutral man

Smiling Man
Vector arithmetic in $z$ space

Glasses man  No glasses man  No glasses woman

Woman with glasses

Radford et al, ICLR 2016
GANs in practice

• Difficult to train (VERY difficult!)
• Difficult to numerically see whether there is progress
  • Plotting the “learning curve” (the minmax objective function) doesn’t help too much
  • Like plotting win rate in P4Bonus (Self-play)
    • Both players get better over time
• Difficult to generate globally consistent structure
• But when GANs work, they work fairly well