Generative Adversarial Nets(Summary)

Introduction

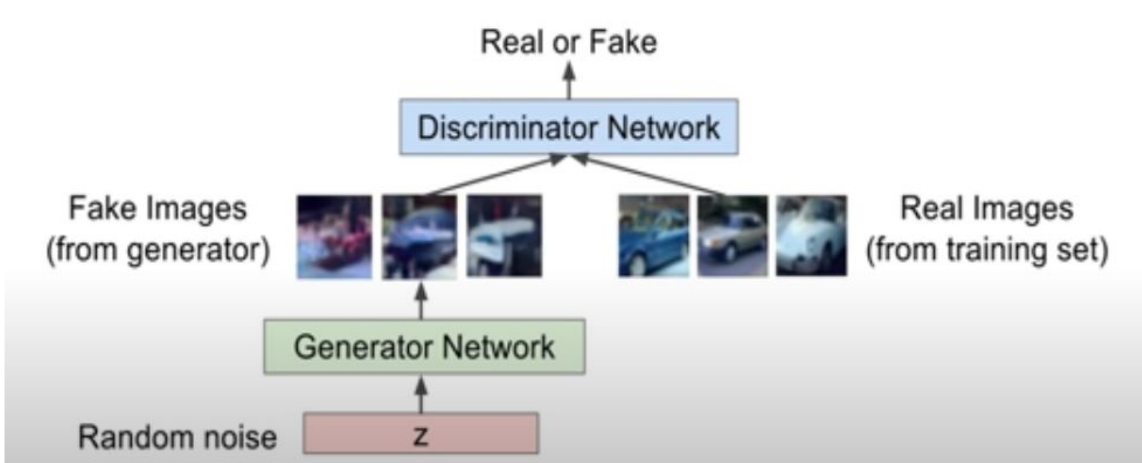
GANs take a game-theoretical approach: learn to generate from 2 player games to come over the problem of the intractable density function.
This model doesn't assume explicit density function as in the case of VAEs.

Adversarial networks

In this, we have two types of networks,

Generator network: try to fool discriminator by generating real looking images.
To learn the generator's distribution p_g over data x, we define a prior on input noise variables $p_g(z)$, then represent a mapping to data space as $G(z; \theta g)$, where G is a differentiable function represented by a multilayer perceptron with parameters g.

Discriminator network: try to distinguish between real and fake images.
It is a second multilayer perceptron $D(x; \theta_d)$ that outputs a single scalar. D(x) represents the probability that x came from the data rather than p_g.



Mathematics behind above intuition

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g}\max_{\theta_d}\left[\mathbb{E}_{x\sim p_{data}}\log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z\sim p(z)}\log(1-\underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}})\right]$$

- Discriminator(θ_d) try to maximize objective function such that D(x) is close to 1(real) and D(G(z)) is close to 0(fake).
  D is trained so that we get the maximum value of log($D_{\theta d}$(x)) that is internally trying to label dataset images as real, also D is maximizing log(1-$D_{\theta d}$($G_{\theta g}$(z)) this internally trying to label generated images as fake images.
- Generator (θ_g) try to minimize objective such that D(G(Z)) is close to 1.
  It is trying to minimize log(1-$D_{\theta d}$($G_{\theta g}$(z)) thus setting θg (through gradient descents) such that it can fool discriminator.
  Rather than training G to minimize log(1 - D(G(z))) we can train G to maximize log(D(G(z)) because it provides steep gradients initially then it provides non steep.

Implementation

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Note: Objective function obtain minimum for a given generator when $p_g = p_{data}$

Disadvantages

- There is no explicit representation of $P_g(x)$
- D must be well sync with G otherwise it could create problem.