Generative adversarial Nets

워낙 유명한 내용이라 수식을 제외한 몇 부분은 생략하기로 한다.

1 Introduction

2 Related work

3 Adversarial nets

- data $\boldsymbol{x}$에 대한 generator의 분포 p_g를 얻기(학습하기) 위해 input noise variables에 대한 prior $p_{\boldsymbol{z}}(\boldsymbol{z})$를 정의한다. Then represent a mapping to data space as $G(\boldsymbol{z};\theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. Also define $D(\boldsymbol{x};\theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ 는 $\boldsymbol{x}$가 실제 data에서 나온 값인지에 대한 확률을 output으로 나온다.

- 목적함수

$$\min_D \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

실제 학습을 진행할 때는 두 네트워크를 동시에 학습시키지 않고 따로따로 업데이트를 한다. D를 학습시킬 때는 G를 고정한 상태에서, G를 학습시킬 때는 D를 고정한 상태에서 진행한다.

먼저, D를 학습하기 위해서는 G의 parameter를 고정한 상태에서 batch size의 수만큼의 data를 넣어서 V를 높이는 방향으로 parameter를 업데이트한다.

$$\begin{aligned}\max_D V(D) &= E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))] \\ &= \frac{1}{m}\sum_{i=1}^{m}\log D(x^i) + \frac{1}{m}\sum_{i=1}^{m}\log(1-D(G(z^i)))\end{aligned}$$

D의 parameter 고정한 상태에서 가짜 데이터 m개를 생성해 V를 계산한 뒤, G에 대한 V의 gradients를 구하고 V를 낮추는 방향으로 G의 parameter를 업데이트한다.

$$\begin{aligned}\min_G V(G) &= E_{z\sim p_z(z)}[\log(1-D(G(z)))] \\ &= \frac{1}{m}\sum_{j=1}^{m}\log(1-D(G(z^j)))\end{aligned}$$

그런데 여기서 문제는 학습 초기에 G가 학습이 되지않는 다는 점이다. G의 목적함수를 보면 $\log(1-x)$의 형태인데 0 근처에서의 기울기가 상당히 작다는 것을 알 수 있다. 학습 초기에는 D가 분류를 잘해서 G에서 만든 것을 쉽게 맞추게 되고 parameter 업데이트가 느릴 수 밖에 없다. 그래서 그 부분을 $\log(x)$로 바꾼다.

$$\begin{aligned}\min_G V(G) &= E_{z\sim p_z(z)}[\log(1-D(G(z)))] \\ &\to -E_{z\sim p_z(z)}[\log D(G(z))]\end{aligned}$$

4 Theoretical Result

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.

- For G fixed(G가 학습이 잘되었다고 가정해야하는 듯), the optimal discriminator D is

$$D_G^*(\boldsymbol{x}) = \frac{P_{data}(\boldsymbol{x})}{P_{data}(\boldsymbol{x}) + P_g(\boldsymbol{x})}$$

  - [proof] 아래의 식을 미분하여 위의 값을 구할 수 있다.

$$\begin{aligned} V(G,D) &= \int_x p_{data}(x) \log(D(x)) dx + \int_z p_z(z) \log(1 - D(g(z))) dz \\ &= \int_x p_{data}(x) \log(D(x)) + p_g(x) \log(1 - D(x)) dx \end{aligned}$$

- 최적의 D가 전제되었다면 gan의 목적함수를 최적화하는 것은 p_{data}와 p_g사이ㅢ Jensen-Shannon divergence를 최소화하는 것과 같다. 즉, data의 분포와 G가 생성하는 분포 사이의 차이를 줄인다고 할 수 있다. 따라서 최적의 결과는 $p_{data} = p_g$이다.

$$\begin{aligned} \min_G V(D^*, G) &= E_{x \sim p_{data}(x)}[\log D^*(\boldsymbol{x})] + E_{z \sim p_g(z)}[\log\{1 - D^*(\boldsymbol{x})\}] \\ &= E_{x \sim p_{data}(x)} \left[\log \frac{P_{data}(\boldsymbol{x})}{P_{data}(\boldsymbol{x}) + P_g(\boldsymbol{x})} \right] + E_{x \sim p_g(z)} \left[\log \frac{P_g(\boldsymbol{x})}{P_{data}(\boldsymbol{x}) + P_g(\boldsymbol{x})} \right] \\ &= \int_x p_{data}(x) \log \left[\frac{P_{data}(\boldsymbol{x})}{P_{data}(\boldsymbol{x}) + P_g(\boldsymbol{x})} \right] dx + \int_x p_g(x) \log \left[\frac{P_g(\boldsymbol{x})}{P_{data}(\boldsymbol{x}) + P_g(\boldsymbol{x})} \right] dx \\ &= -\log 4 + \int_x p_{data}(x) \log \left[\frac{p_{data}(x)}{\frac{p_{data}(x) + p_g(x)}{2}} \right] dx + \int_x p_g(x) \log \left[\frac{P_g(\boldsymbol{x})}{\frac{P_{data}(\boldsymbol{x}) + P_g(\boldsymbol{x})}{2}} \right] dx \\ &= -\log 4 + KL\left(p_{data}(x) || \frac{p_{data}(x) + p_g(x)}{2} \right) + KL\left(p_g(x) || \frac{p_{data}(x) + p_g(x)}{2} \right) \\ &= -\log 4 + 2 * JS\left(p_{data}(x) || p_g(x) \right) \quad JS(\cdot) = 0 \text{ if } p_{data} = P_g \end{aligned}$$

5 Experiments

6 Advantages and disadvantages

- advantage
  - No need of Markov chain only backprop with gradient.
  - No need of inference during training.
  - Wide variety of functions can be incorporated into the model.
- disadvantage
  - There is no explicit rep[resentation of $p_g(\boldsymbol{x})$.
  - D must be synchronized well with G during training.

7 Conclusions and future work