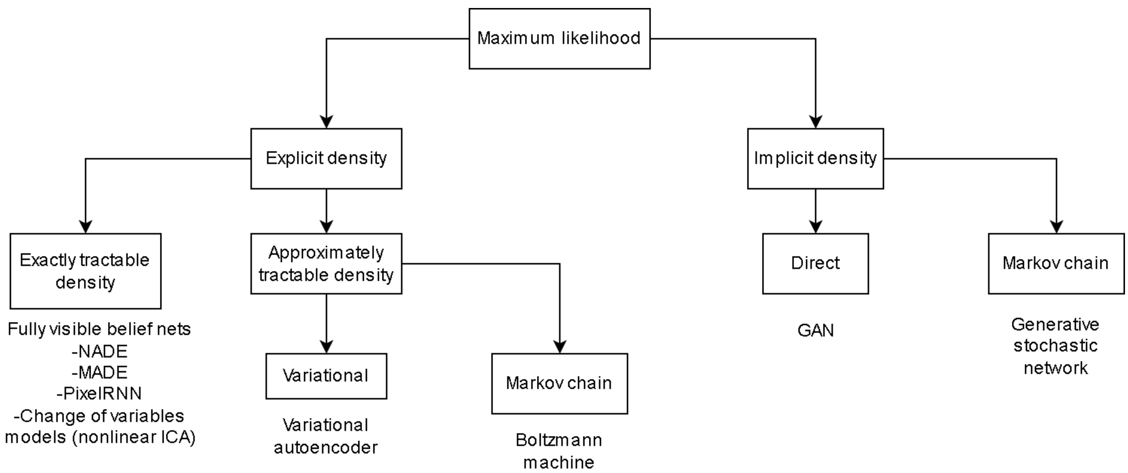
GAN(Generative Models) : cs231n-13

`2022년 11월 28일 작성`

Generative Adversarial Networks (GAN)

개요

- GAN이외의 생성모델에서는 explicit density function을 계산하고 그 값을 최대화하는 방향으로 정의했지만, 이제는 데이터를 단순히 샘플링 하는 것으로 정의해 보자는 것이 **GAN** 의 출발



- 구체적인 식이 없기 때문에 **게임이론**의 **minimax** 적용 : 실제 data의 distribution을 모방하려는 generative model G와, 입력된 data가 (G로부터 생성된 fake data가 아닌) 실제 data일 확률을 출력하는 discriminative model D가 서로 **adversarial process**를 통해 G를 학습하는 방법을 제안
    - G (generator)는 training data의 distribution을 (모방하여) 생성하는 것이 목표 (즉, 생성한 이미지를 D가 참으로 인식할 수 있도록 최대한 진짜처럼 만들어내는 것!)
    - D (discriminator)는 실제 training data sample과 생성된 data sample을 구분하지 못해 항상 0.5를 출력하는 것이 목표

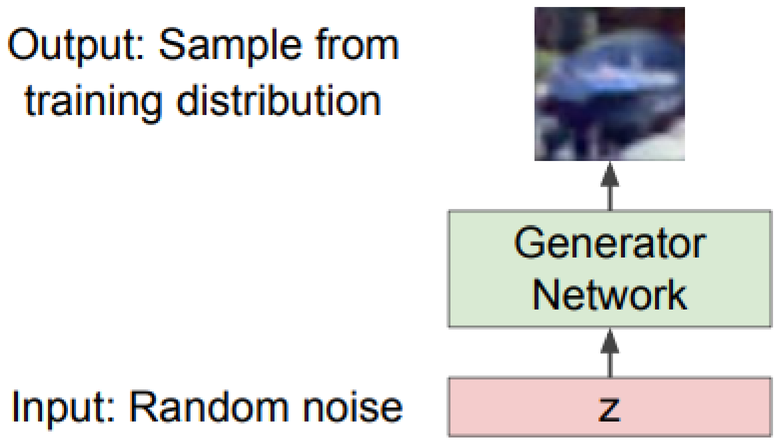
문제

1. **문제** : complex, high-dimensional training distribution으로부터 데이터를 샘플링 하고자 함. 그러나 직접적인 방법이 없음 (왜냐하면, 구체적인 밀도함수를 정의하지 않았기 때문)
2. **해결** : 정규분포(강의에선 simple distribution이라 표현, e.g. random noise)로부터 데이터를 랜덤 샘플링 하여 타겟 분포로 보내는 함수를 생각하고, 함수의 이미지가 타겟 분포와 흡사해지도록 함수를 학습시키자!

질문

1. 어떻게 하면 정규분포로부터 랜덤추출한 데이터의 함수값에 대한 분포를 타겟 분포에 직접적으로 잘 근사할 수 있을까?
2. 해결과정
    a. **원하는 것** : 그림과 같이 input($z \sim N(0,1)$, random noise)값이 output(sample from training distribution)으로 잘 mapping되는 함수를 근사하고 싶음

    (즉, 이미지/영상이라면 각 픽셀값이 잘 mapping되는 함수를 찾고 싶은 것임)



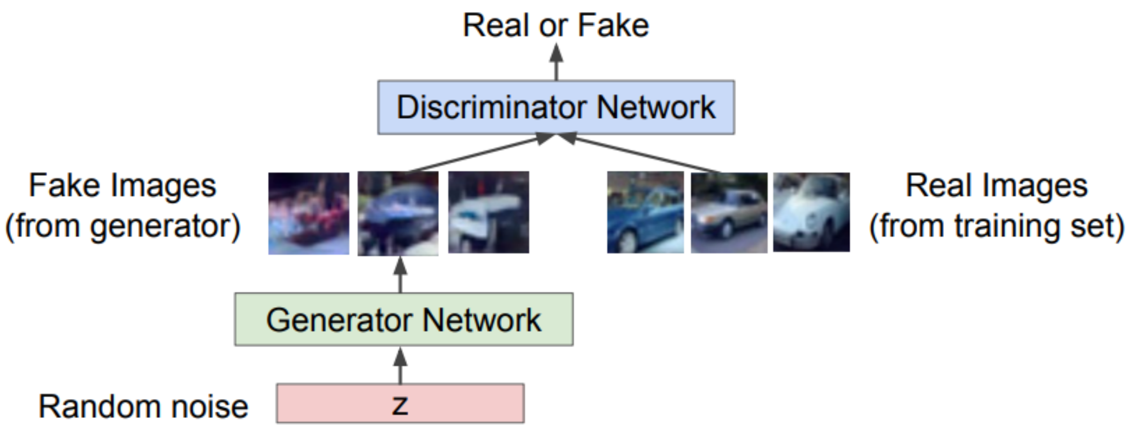
(cs231n-Lecture-13-p.103)

2. 그 과정이 너무 복잡하니 **neaural network**를 이용하여 근사하자!
2. 그래서 **loss function**을 정의해야 할 필요성 인식 (그래야 파라미터들을 업데이트 하며 학습 진행할 수 있음)
3. **loss function**은 어떻게 정의할 것인가?

GAN의 학습 원리 : Two-player game

- **<u>Generator network (G)</u>** : generator가 실제 이미지처럼 만들어낸 이미지를 discriminator가 진짜라고 믿게 하는 것이 목표
- **<u>Discriminator network (D)</u>** : 실제 이미지와 가짜 이미지를 잘 판별하는 것이 목표

1. Two-플레이어 게임 이론 : minimax game으로 함께 훈련



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

2. 목적함수 (Minimax objective function)

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}(x)} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - d_{\theta_d}(G_{\theta_g}(z))) \right]$$

- $\theta_d, \ \theta_g$: the parameter sets in discriminator and generator, respectively
- 규칙 : 참=1, 거짓=0, $D(real) \in (0, 1)$
- 목표 (minimax 적용) : D는 목적함수를 maximize for θ_d (즉, 괄호 안을 0에 가깝게), G는 목적함수를 minimize for θ_g (즉, 괄호 안을 $-\infty$에 가깝게)
    - D maximizes the objective function : **Gradient ascent** on D for θ_d

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

        - D는 real data x (ground truth)를 참(1)으로 인식
          즉, $D_{\theta_d}(x) \sim 1 \Rightarrow \log D_{\theta_d}(x) \sim 0$
        - D는 generated fake data $G(z)$를 거짓(0)으로 인식
          즉, $D_{\theta_d}(G_{\theta_g}(z)) \sim 0 \Rightarrow \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \sim 0$

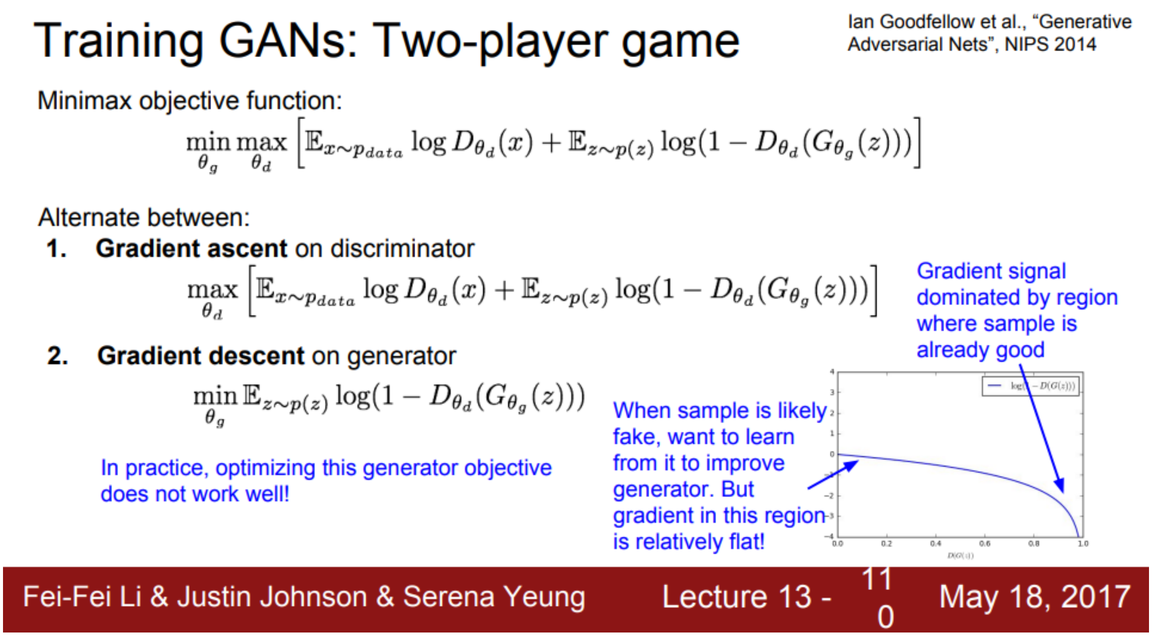
- G minimizes the objective function for fixed parameters θ_d : **Gradient descent** on G for θ_g

$$\min_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

  - D는 generated fake data $G(z)$를 참(1)으로 인식하도록 G 학습

    즉, $D_{\theta_d}(G_{\theta_g}(z)) \sim 1$

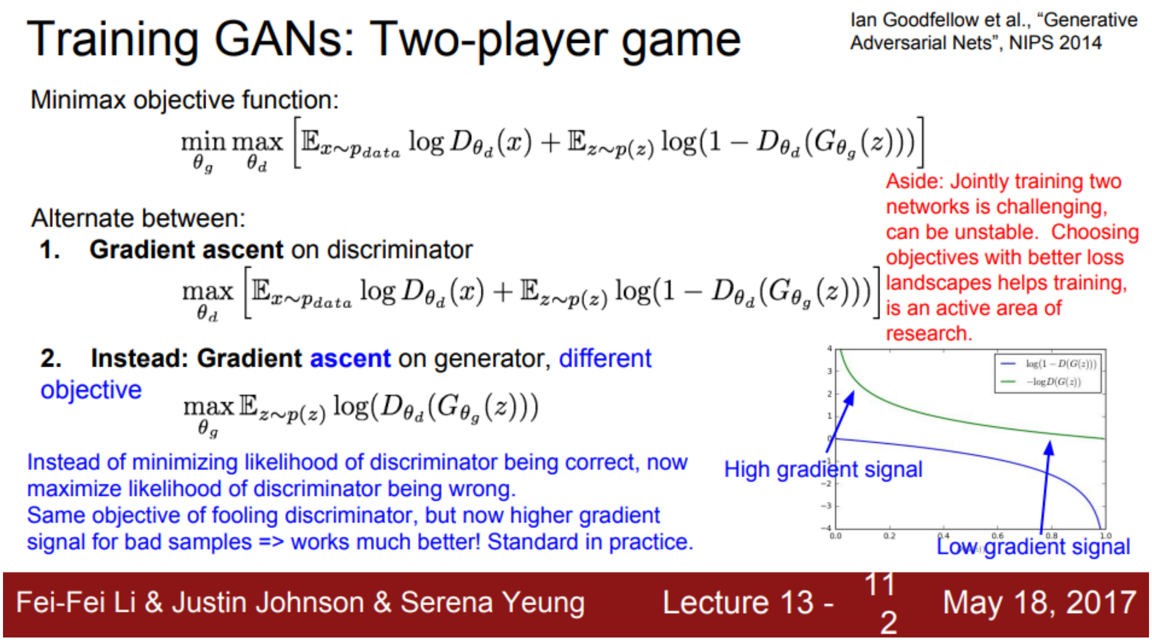
- 문제점 : 실제 학습 시, generator objective의 optimizing 잘 안 됨



  - 우측 하단의 그래프에서 오른쪽으로 하강하는 부분은 학습이 잘 되는 곳에서 경사가 가파름. 즉, 학습 속도가 너무 빠름
  - 반면 왼쪽의 완만한 곳(기울기가 작은 곳)은 학습 속도가 느린데, 가짜 이미지가 잘 판별되는 곳이므로 오히려 빠른 학습이 필요함
  - 즉, 그래프 좌측은 기울기 가파르게, 우측은 완만하게 해야 함

- 해결 방법 : Instead. **Gradient ascent** on G, **different objective** !!

$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$



- 두 번째 objective function을 위의 녹색 함수로 바꾼 후 graidient ascent를 적용하면, 녹색 그래프의 좌측에선 학습 빨라지고(즉, step이 커짐), 우측에선 학습이 느려짐 (즉, step 작아짐)

- **Aside** : (1) D, G를 동시에 학습시키면 불안정할 수 있고, (2) 좀 더 나은 objective function을 찾는 것도 활발한 연구 분야 중 하나
  - 처음 제시된 objective function은 vanilla GAN이라는 2014년 논문에 처음 등장
  - 저 함수의 문제점은, 모델이 생성한 이미지가 일그러지는 "model collapse"라는 현상이 자주 발생하는 것
  - WGAN : 위의 문제점을 해결하고자, 좀 더 안정적이고 훈련이 잘 되는 새로운 objective function 적용 (이후에 WGAN-GP라는 모델 등장)

3. 학습 과정 (GAN training algorithm)

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.

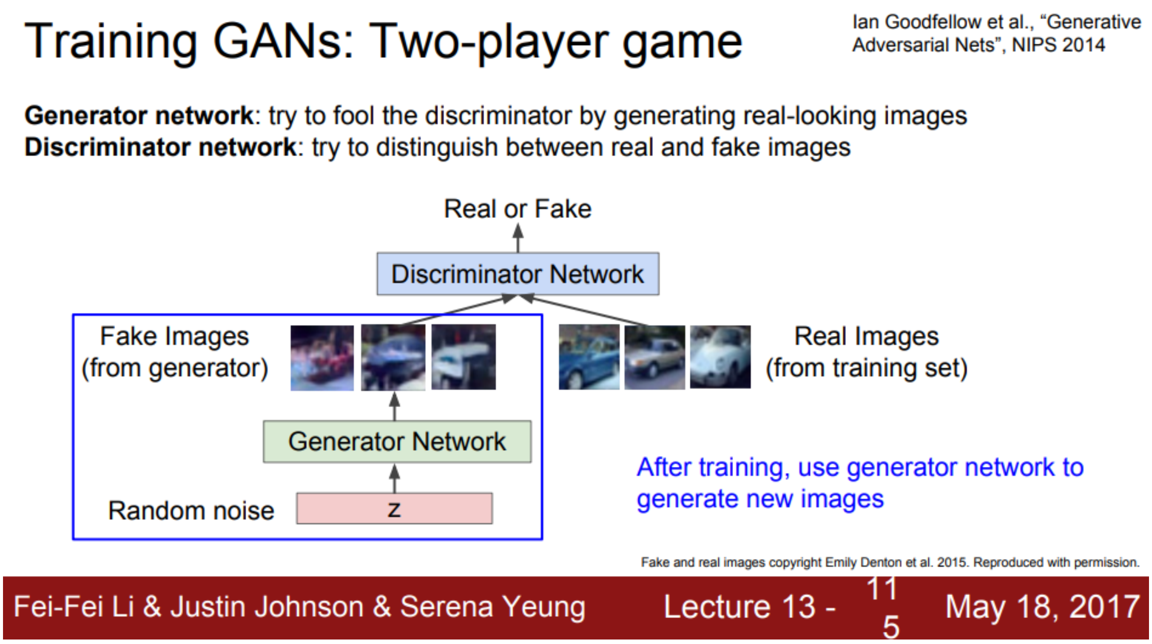
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

- **for** the number of training iteration **do** = per $1 - epoch$
- k step : 배치사이즈 조정, k=1인 경우 좀 더 안정적이나 커도 상관 없음
- **for** k step **do** : 이 단계에서 D의 학습
  - (normal or uniform) prior distribution으로부터 미니배치 수만큼 뽑은 m개의 노이즈 샘플(input data)로 미니배치 샘플 구성
  - 실제 training dataset으로부터 m개의 실측데이터(ground truth) 추출하여 미니배치 샘플 구성
  - Stochastic Gradient Ascent에 의한 D의 파라미터 업데이트 (k번 반복 진행)
- **end for** : 이 단계에서 G의 학습
  - noise prior distribution으로부터 m개의 샘플 선택하여 배치 구성
  - Stochastic Gradient Ascent에 의한 G의 파라미터 업데이트 (단, D의 파라미터는 위에서 학습시킨 결과를 고정시킨 후 G 학습)
- **end for** : 여기까지 한 번 돌아가는 게 $1 - epoch$

💡 D, G 각각의 objective function이 다르므로 파라미터 업데이트는 동시에 진행하지 않고 D를 우선 진행한 후, G 진행

4. 학습 후

- 학습이 완료된 G를 이용하여 새로운 이미지 생성



- *Figure 2, 3*은 MNIST, Toronto Face Database (TFD), CIFAR-10으로 학습한 GAN의 generator가 생성한 데이터들의 예
- 노란색 박스 = ground truth 이미지, 나머지 부분 = gt image와 비슷하게 생성한 이미지
- 문제점 : 생성된 이미지가 여전히 blurry

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

DCGAN - Radford et al, ICLR 2016

1. Convolutional Architectures

- 초기 GAN 모델 : D, G 모두 MLP(Multi layer perceptron) 사용
- 이미지를 다루다 보니 CNN의 필요성 인식
- 문제는 이미지가 conv. net을 통과하는 순간 사이즈가 줄기 때문에 이걸 다시 키울 수 있는 장치(convolutional transpose)가 필요하고 이것을 적용한 모델이 **DCGAN** (Deep Convolutional GAN)
    - G : 부분적으로 strided convolution을 사용하는 upsampling 네트워크

- D : convolution 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.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

- stable **DCGAN**을 위한 가이드라인
    a. 모든 pooling layer를 strided conv. (discriminator)와 frational-strided conv. (generator)로 교체, $stride \geq 2$
    b. G, D 모두 $batchnorm$ 사용
    c. 깊은 구조의 fully connected hidden layer들 제거
    d. G에서 Tanh로 출력되는 것을 제외한 모든 layer에 대하여 ReLU 사용
    e. D의 모든 layer에서 $\mathrm{LeakyReLU}$ 사용
- (우측 → 좌측) 과정이 conv.이었고 이 반대 과정이 필요, 이걸 적용한 모델이 **DCGAN**

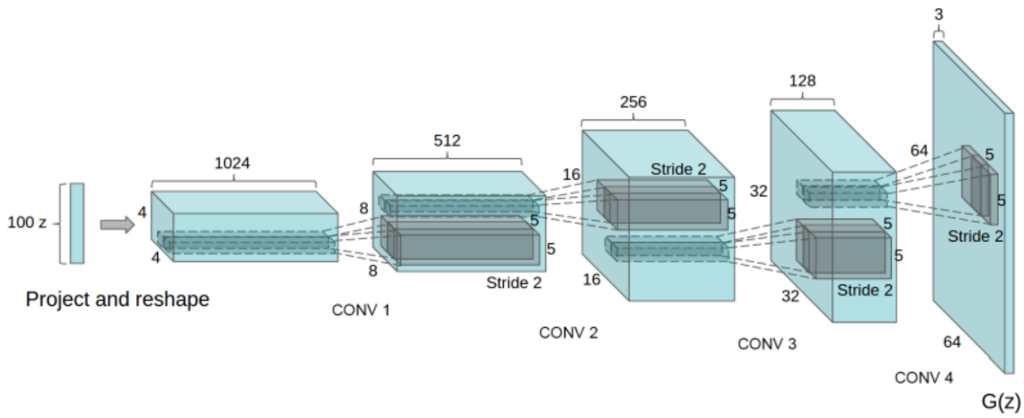
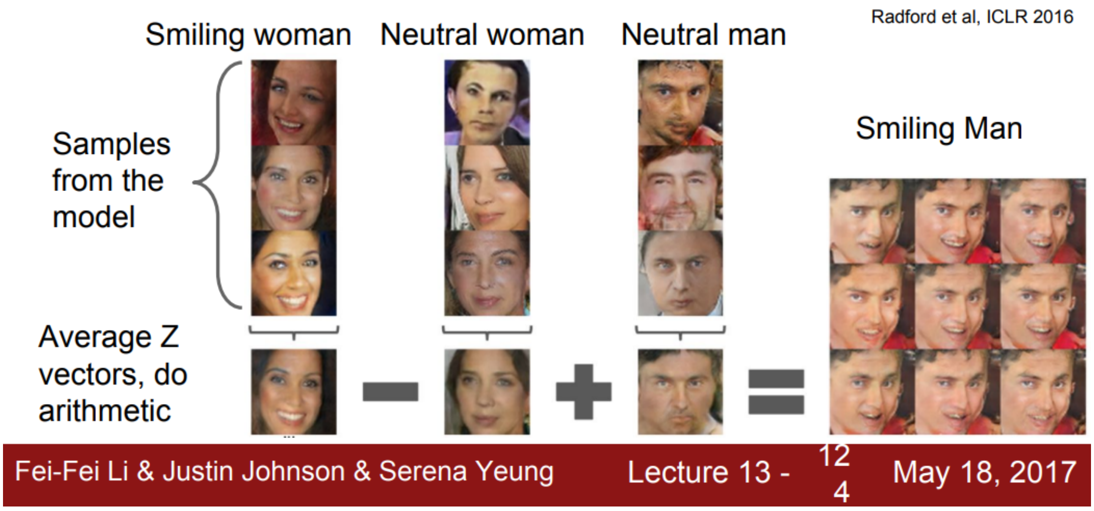


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

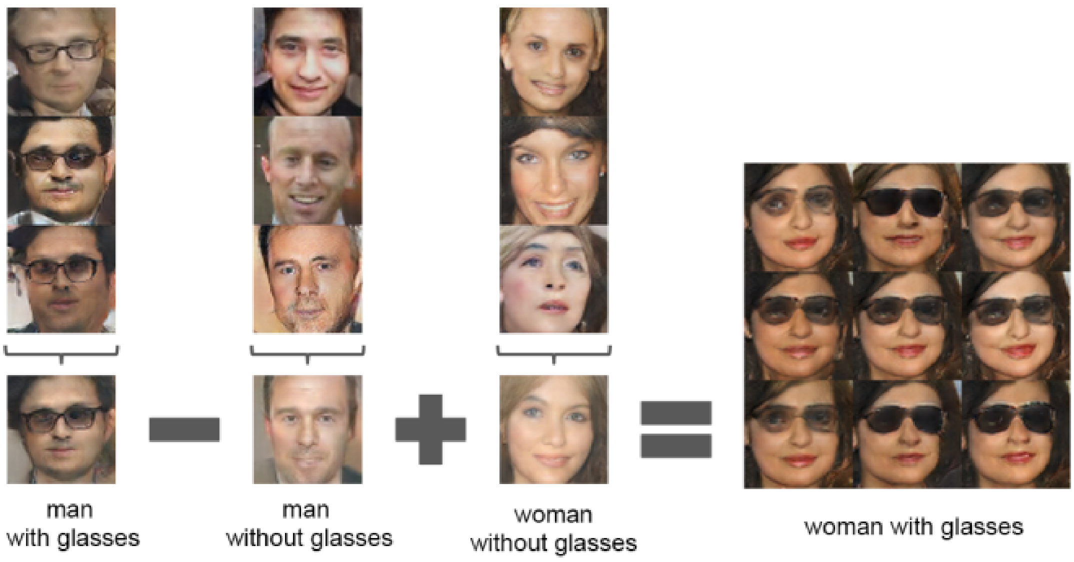
Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

2. Interpretable Vector Math

- vector("King") - vector("Man") + vector("Woman") resulted in a vector whose nearest neighbor was the vector("Queen")
- 같은 특성은 갖는 샘플들을 하나의 열벡터로 보고 그것들의 평균 벡터 Z 추출
  - **Smiling ~~Woman~~ - ~~Neutral~~ ~~Woman~~ + ~~Neutral~~ Man = Smiling Man**



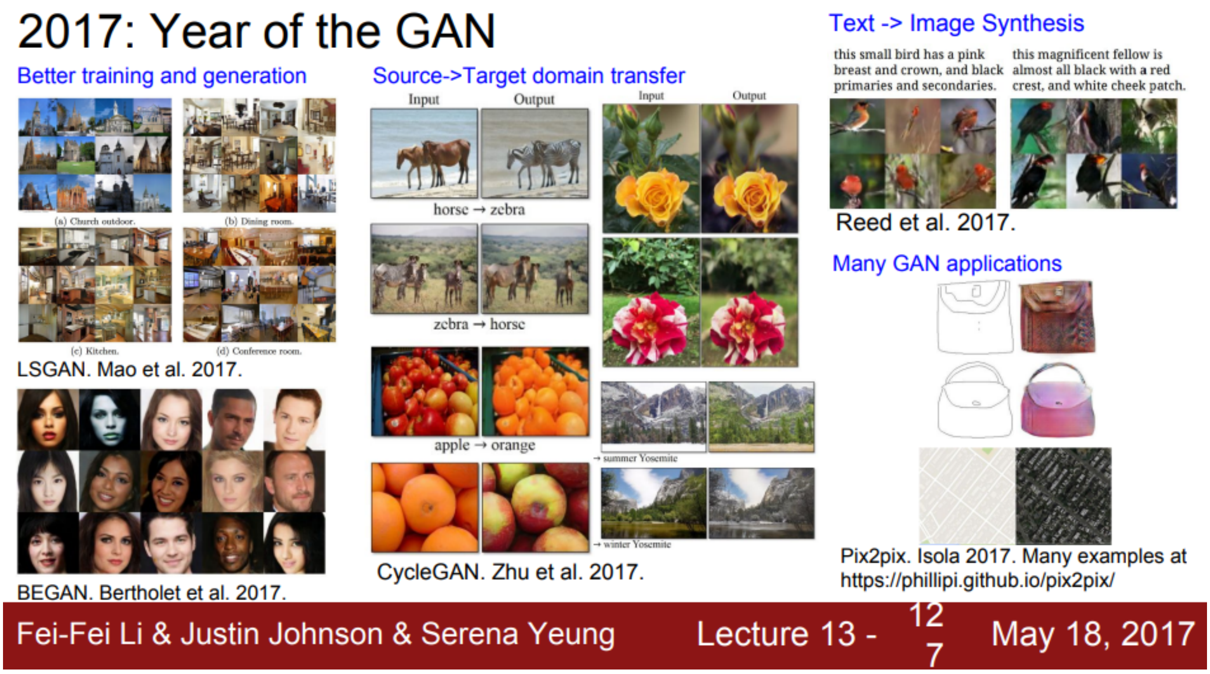
  - **~~Man~~ with Glasses - ~~Man~~ ~~without Glasses~~ + Woman ~~without Glasses~~ = Woman with Glasses**

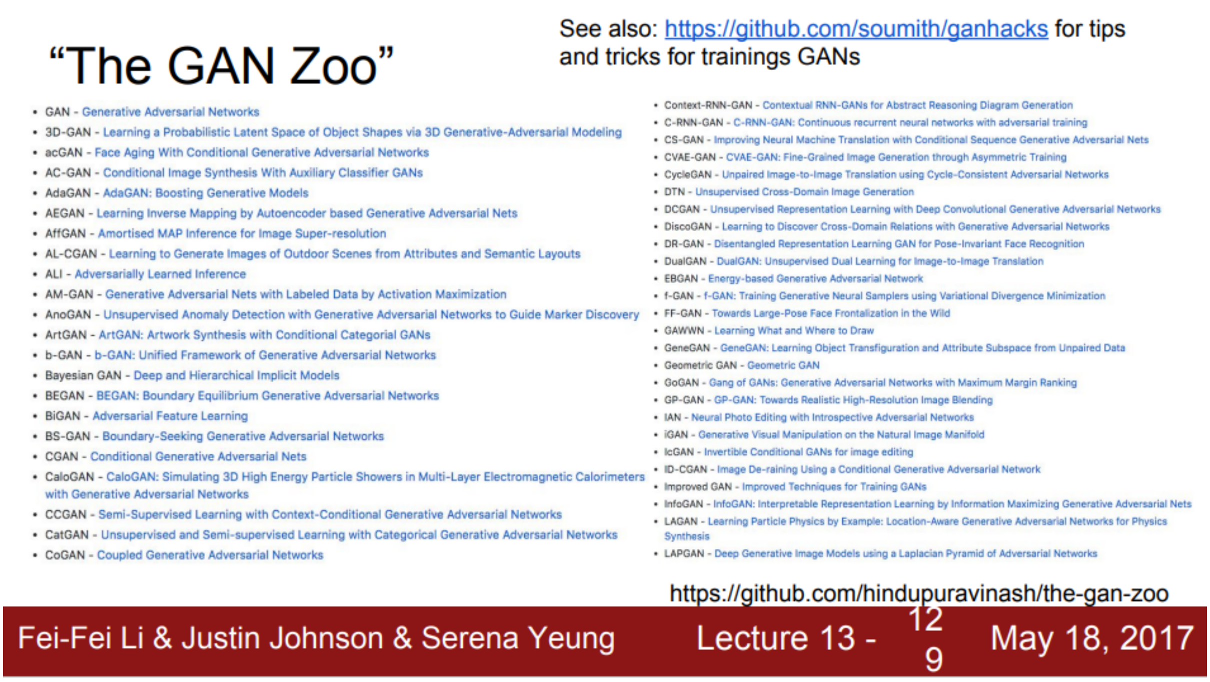


3. 2017년 이후

- LSGAN, Mao et al. 2017.
- BEGAN, Bertholet et al. 2017.

- cycleGAN, Zhu et al. 2017.
- text to image, Reed et al. 2017.
- many GAN applcations : pxi2pix, Isola 2017. …





https://github.com/soumith/ganhacks

4. 요약

1. 최적화는 구체적인 밀도 함수로 작동하지 않음, 정규분포를 따르는 샘플 벡터를 바로 타겟 이미지로 보내는 directly mapping 자체를 신경망으로 학습시킴
2. 구체적인 목적 함수가 없기 때문에 게임이론 적용
3. 장점 : 너무 잘 됨
4. 단점 :
    - 학습하기 어려움, unstable : 하이퍼파라미터에 따라 변화 심하고, 데이터 수량이 부족하면 학습 안 되고, collapse되는 현상 발생
    - $p(x), p(z|x)$와 같은 것을 inference queries로 해결할 수 없음
5. 연구 분야 :
    - Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
    - Conditional GANs, GANs for all kinds of applications

References

1. Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
2. Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016
3. Jie Gui et al, “A Review on Generative Adversarial Networks : Algorithms, Theory, and Applications”, IEEE 2021

그 외 참고

- cs231n lecture 13