PR12와 함께 이해하는

GANs

Jaejun Yoo
Ph.D. Candidate @KAIST
PR12
16th Apr, 2017

* Generative Adversarial Nets Ian Goodfellow et al. 2014를 바탕으로 작성한 리뷰

Generative Adversarial Network

Generative Adversarial Network

PREREQUISITES

Generative Models

"FACE IMAGES"

PREREQUISITES

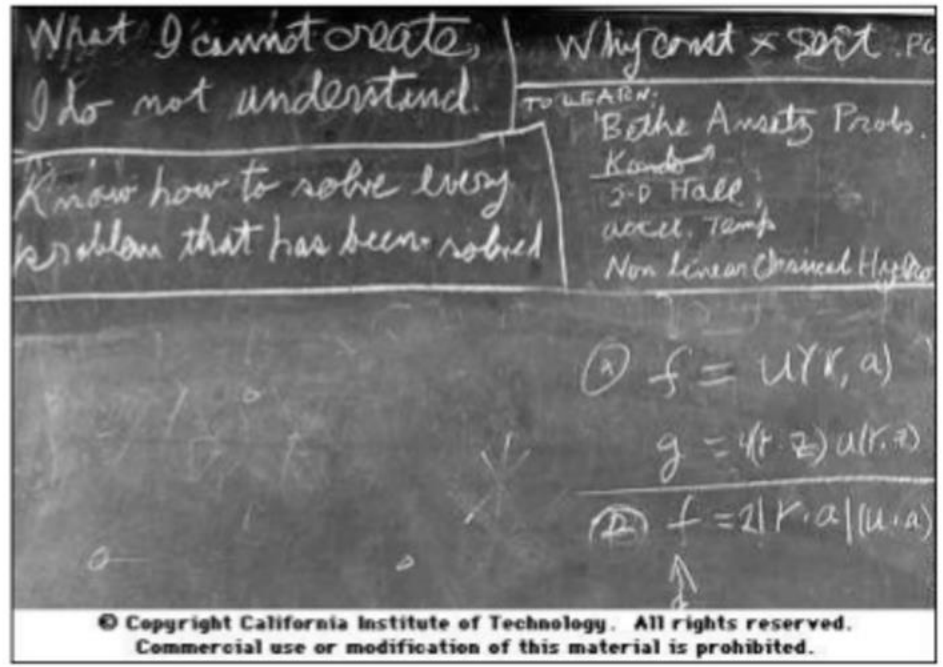
Generative Models

Generated Images by Neural Network

* Figure adopted from ***BEGAN*** paper released at 31. Mar. 2017
David Berthelot et al. Google (link)

PREREQUISITES

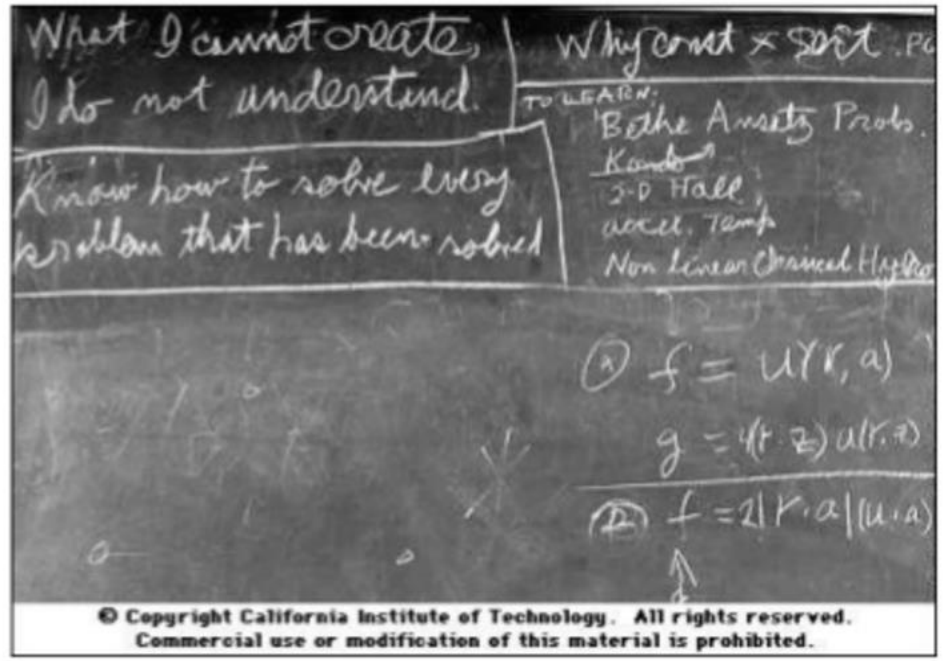
Generative Models



"What I cannot create, I do not understand"

PREREQUISITES

Generative Models

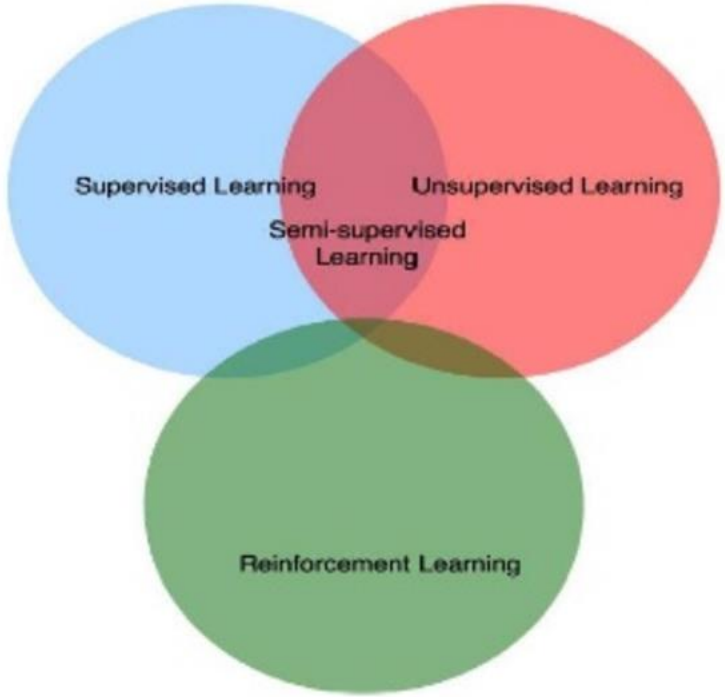


"What I cannot create, I do not understand"

If the network can learn how to draw cat and dog separately,
it must be able to classify them, i.e. feature learning follows naturally.

PREREQUISITES

Taxonomy of Machine Learning



From **David silver**, Reinforcement learning (UCL course on RL, 2015)

- **"Pure" Reinforcement Learning** (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning** (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning** (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- **(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)**

From **Yann Lecun**, (NIPS 2016)

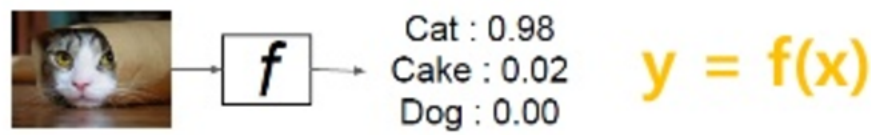
PREREQUISITES

Clip slide

Introduction

Supervised Learning

- More flexible solution
  - Get probability of the label for given data instead of label itself



kakaobrain

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

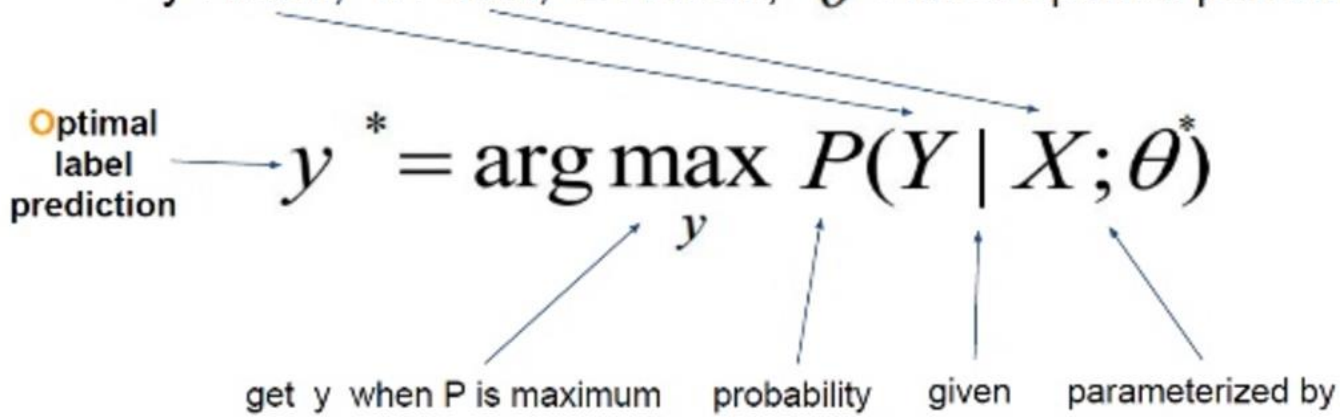
PREREQUISITES

Clip slide

Introduction

Supervised Learning

- Mathematical notation of **classifying** (greedy policy)
  - y : label, x : data, z : latent, θ^*: fixed optimal parameter

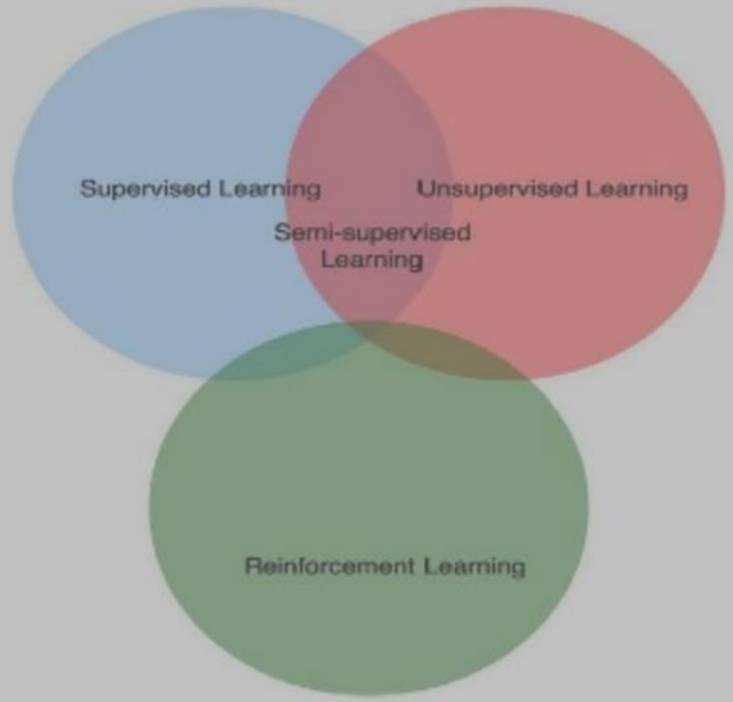


kakaobrain

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

PREREQUISITES

Taxonomy of Machine Learning



From **David silver**, Reinforcement learning (UCL course on RL, 2015)

- **"Pure" Reinforcement Learning** (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning** (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning** (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- **(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)**

From **Yann Lecun**, (NIPS 2016)

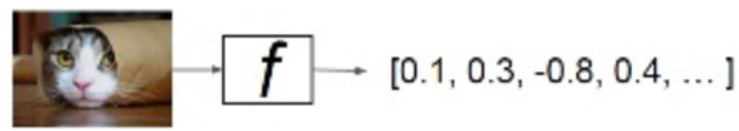
PREREQUISITES

Clip slide

Introduction

Unsupervised Learning

- Find deterministic function ***f*** : $z = f(x)$, x : data, z : **latent**



kakaobrain

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

PREREQUISITES

Introduction

Unsupervised Learning

- More challenging than supervised learning :
  - No label or curriculum → self learning

- Some NN solutions :
  - Boltzmann machine
  - Auto-encoder or Variational Inference
  - Generative Adversarial Network

kakaobrain

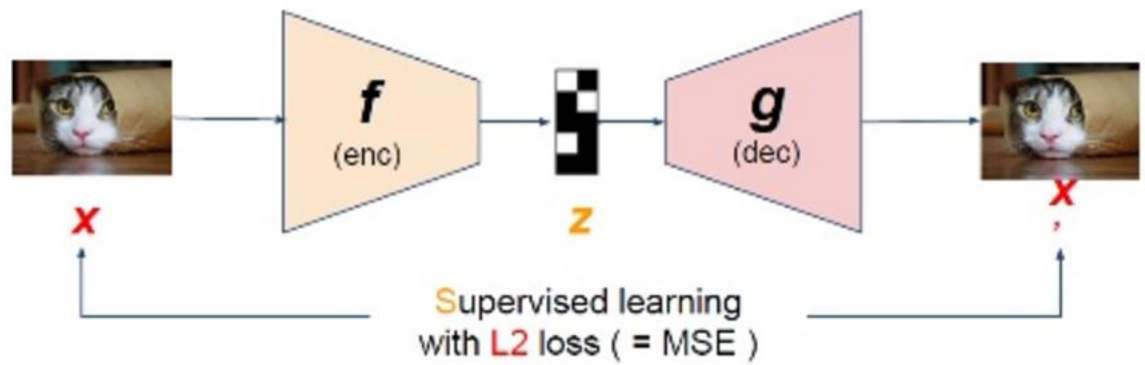
Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

PREREQUISITES

Autoencoders

Stacked autoencoder - SAE

- Use data itself as label → Convert UL into reconstruction SL
- $z = f(x)$, $x = g(z)$ → $x = g(f(x))$
- https://github.com/buriburisuri/sugartensor/blob/master/sugartensor/example/mnist_sae.py



kakaobrain

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

PREREQUISITES

Clip slide

Autoencoders

Variational autoencoder - VAE

- Kingma et al, “Auto-Encoding Variational Bayes”, 2013.

- Generative Model + Stacked Autoencoder
  - Based on **Variational approximation**

Variational approximations Variational methods define a lower bound

$$\mathcal{L}(\boldsymbol{x};\boldsymbol{\theta}) \leq \log p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta}). \tag{7}$$

원하는 이모형이 이것이지만, 우리가 알수 있는 모형으로 하한을 계산함. 이 하한을 높이는 방식

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

PREREQUISITES

Clip slide

Autoencoders

Variational autoencoder - VAE

- Kingma et al, "Auto-Encoding Variational Bayes", 2013.
- Generative Model + Stacked Autoencoder
  - Based on **Variational approximation**

모르는 Z값을 주어졌을때 이미지를 생성하는 디코더가 되고, 이미지를 주어졌을때 Z을 만들어내는 인코더가 됨.
Auto-encoder 라고 해서 학습을 할 수 있고, 문제가 있는데

KL divergence , Likelihood 모델로 이루어졌고, regulazation + Likelihood로 설명할 수 있음.

Variational approximations Variational methods define a lower bound

$$\tilde{\mathcal{L}}^B(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \frac{1}{L}\sum_{l=1}^{L}(\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})) \quad (7)$$

where $\mathbf{z}^{(i,l)} = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(i,l)}, \mathbf{x}^{(i)})$ and $\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

- KL Divergence : https://ko.wikipedia.org/wiki/%EC%BF%A8%EB%B0%B1-%EB%9D%BC%EC%9D%B4%EB%B8%94%EB%9F%AC_%EB%B0%9C%EC%82%B0

PREREQUISITES

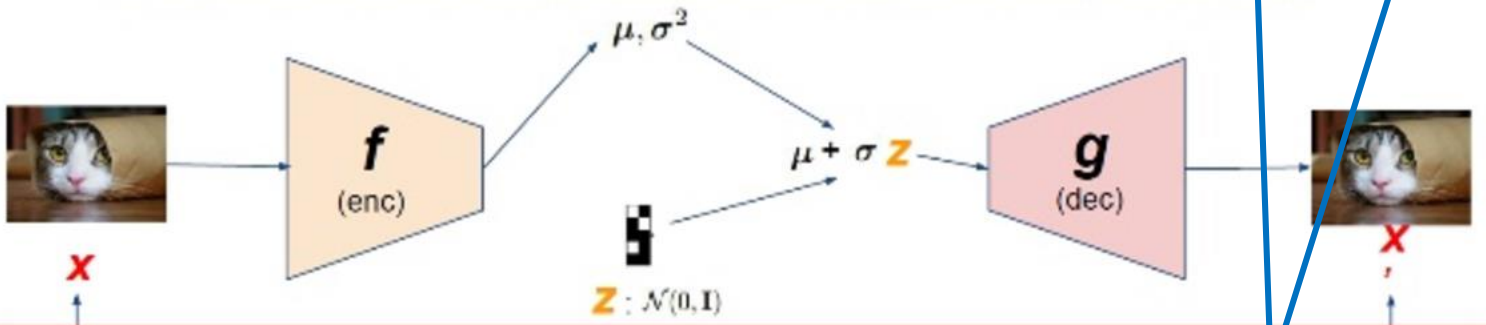
Clip slide

Likelihood부분에서 로직스틱 regression과 비슷하고, 최소값을 학습하고 불러한 경향이 있음.

Autoencoders

Variational autoencoder - VAE

- Training
- https://github.com/buriburisuri/sugartensor/blob/master/sugartensor/example/mnist_vae.py



$$\tilde{\mathcal{L}}^B(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\boldsymbol{\theta}}(\mathbf{z})) + \frac{1}{L}\sum_{l=1}^{L}(\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}))$$

where $\mathbf{z}^{(i,l)} = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(i,l)}, \mathbf{x}^{(i)})$ and $\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

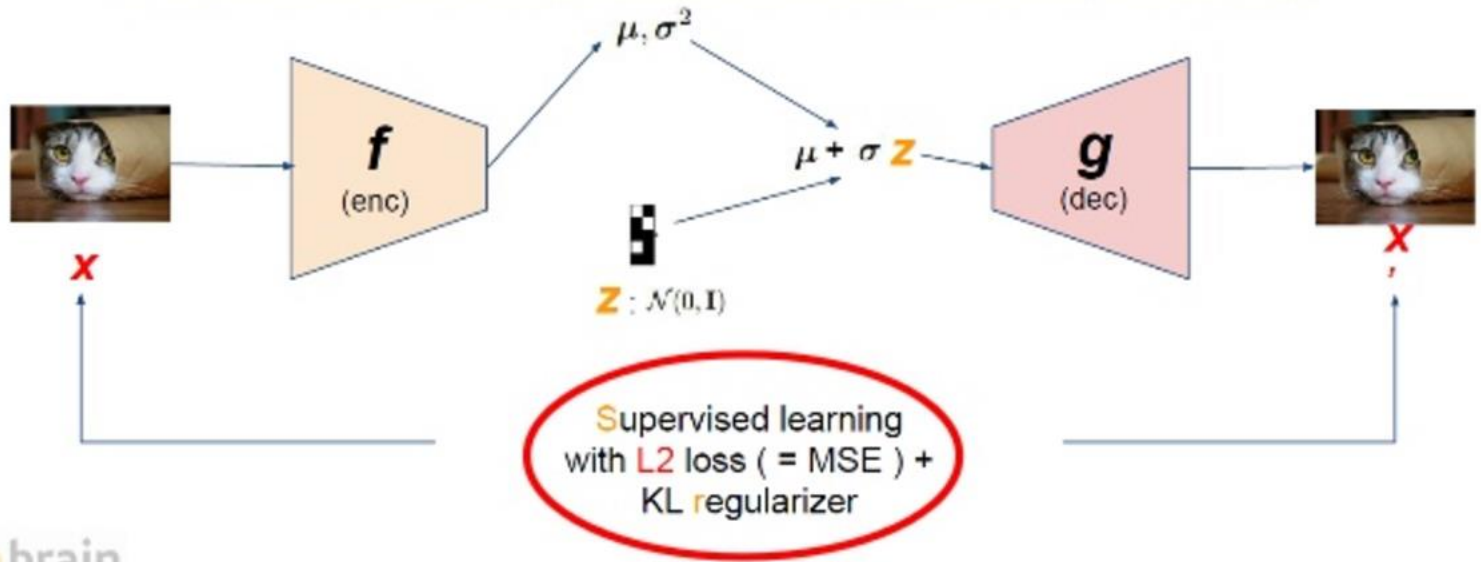
PREREQUISITES

Clip slide

Autoencoders

Variational autoencoder - VAE

- Training
- https://github.com/buriburisuri/sugartensor/blob/master/sugartensor/example/mnist_vae.py



kakaobrain

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

PREREQUISITES

Clip slide

Autoencoders

Variational autoencoder - VAE

- Results

kakaobrain

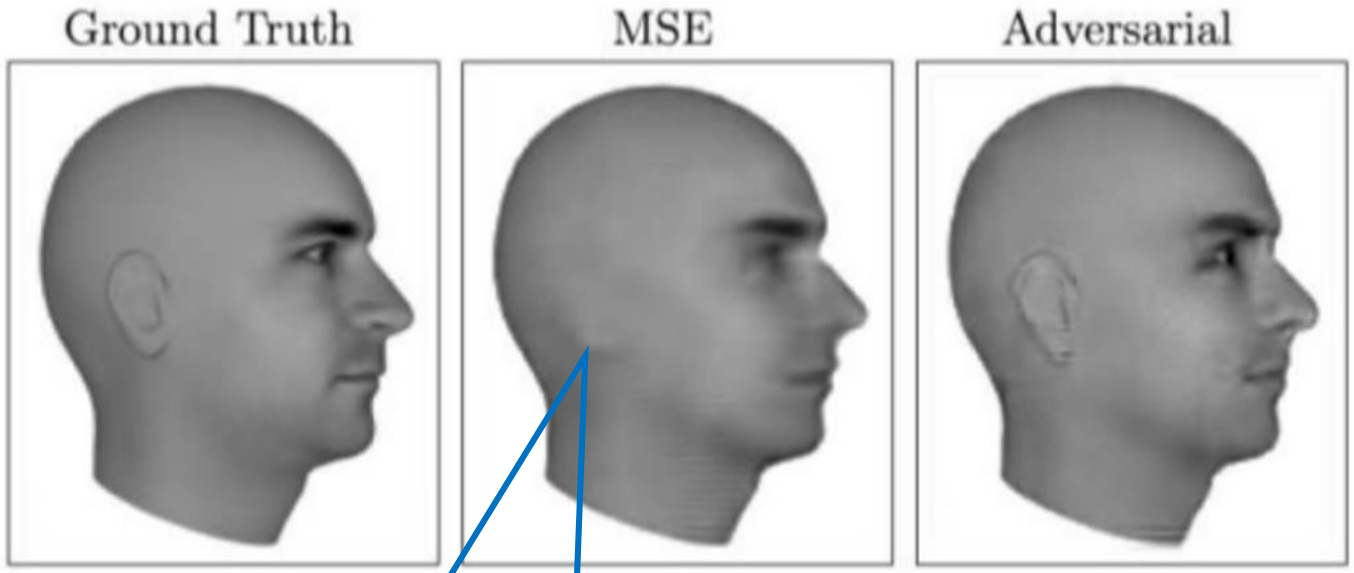
Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

PREREQUISITES

Clip slide

Autoencoders

Variational autoencoder - VAE



Adversarial 은 그중에서 하나만 선택하기 때문에 우리가 짱이라고 함. 왜 그렇게 샤프한 이미지를 줄 수 있냐를 이런식으로 설명함.

MSE 방식의 autoender 는 귀가 없어짐.

kakaobrain

Slide adopted from **Namju Kim**, Kakao brain (SlideShare, AI Forum, 2017)

* Figure adopted from NIPS 2016 Tutorial: GAN paper, Ian Goodfellow 2016

Generative Adversarial Network

Clip slide

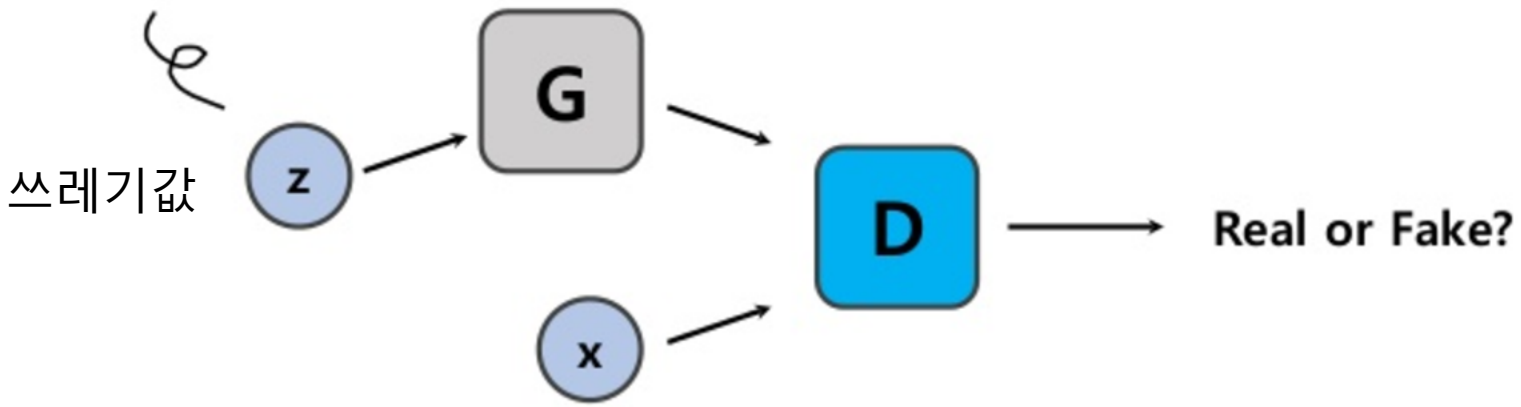
Generative Adversarial Network

SCHEMATIC OVERVIEW

Diagram of Standard GAN

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

Gaussian noise as an input for G

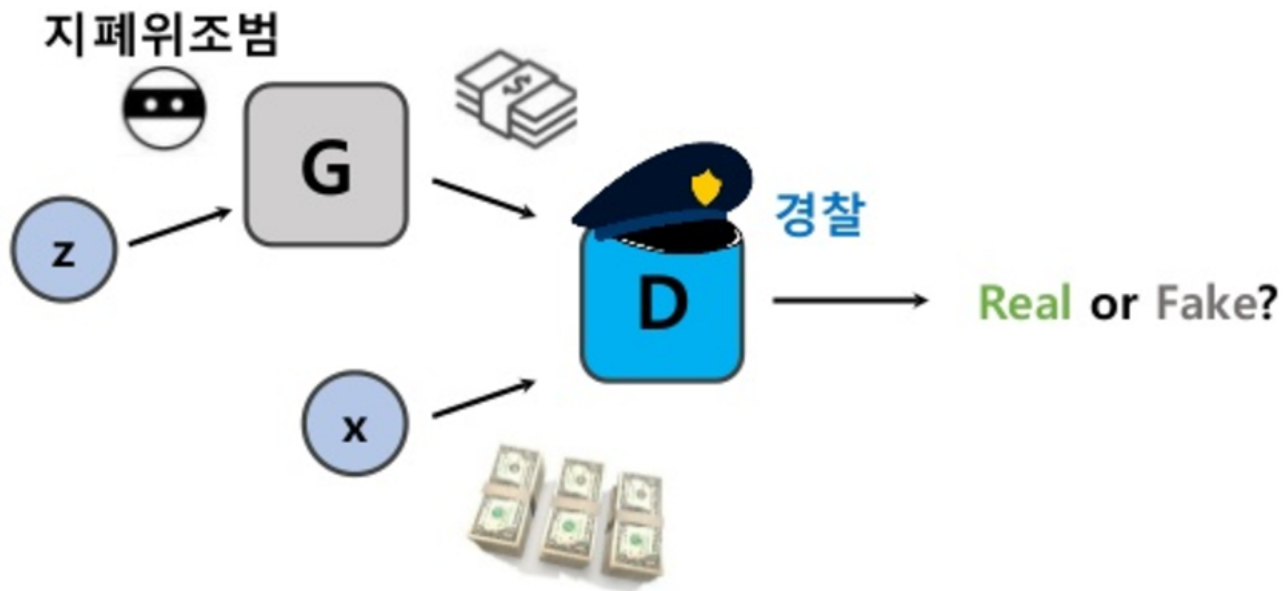


SCHEMATIC OVERVIEW

Clip slide

Diagram of Standard GAN

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

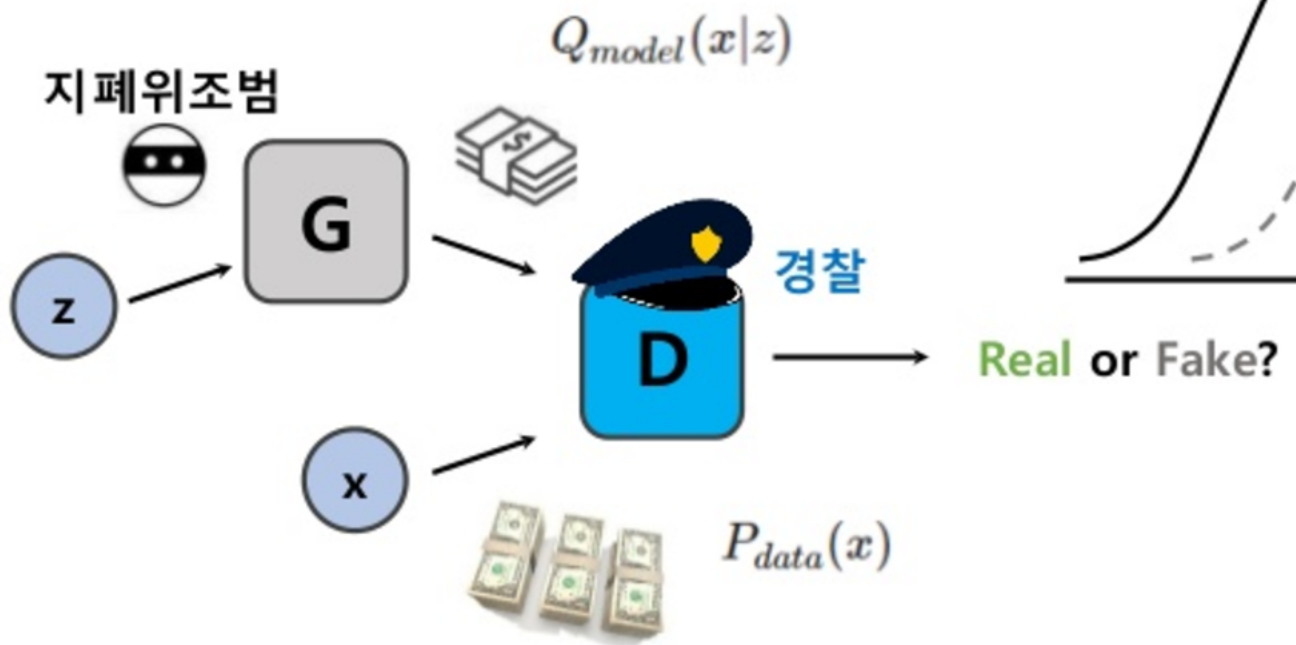


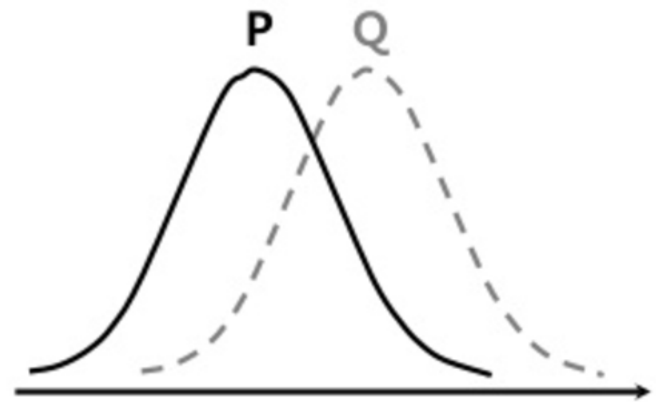
SCHEMATIC OVERVIEW

Clip slide

Diagram of Standard GAN

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$





어떤 노이즈에서 x 이미지를 내보내는 모델을 정의하고

X는 데이터 분포가 있고,

이 실제 데이터 분포 P 를 모르니깐 Q 모델로 어떻게 하면 P에 가깝게 할수 없을까 고민하는 방법

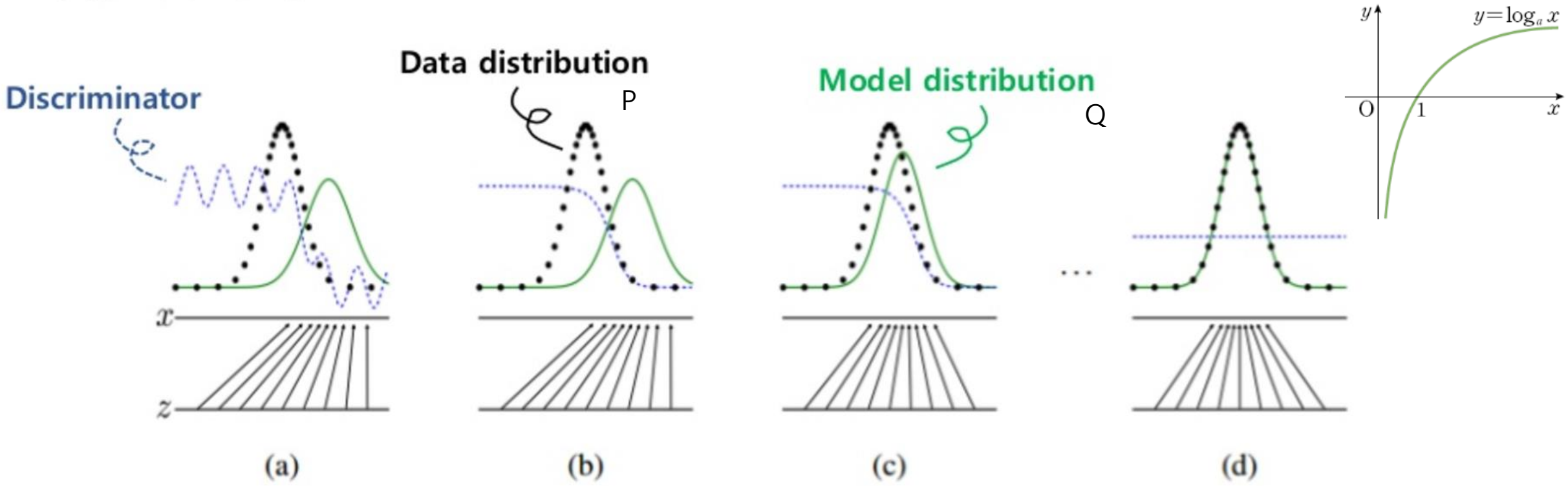
SCHEMATIC OVERVIEW

Clip slide

- 수식에서 D(x)가 실제 이미지면 왼쪽항이 0이 되고, 오른쪽항에서 G(z)가 생성한 이미지를 정확히 하면
- D(G(x)) 가 0 이 되어 V(D, G)의 최대값은 0 이 됨.
- D는 최대값을 갖게 하는 것이 좋은 것이고, G는 이미지와 구분을 못하게 하는것이 좋은것이기 때문에 최소값을 갖도록 학습.

Diagram of Standard GAN

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))]$$



- 처음에는 Q와 P을 쉽게 구분하고, 학습이 진행됨에 따라서 Q가 P에 근접에서 하게 되고 마지막에는 Q와 P을 ½ 확률로 분류하게 됨.

* Figure adopted from Generative Adversarial Nets, Ian Goodfellow et al. 2014

THEORETICAL RESULTS

Clip slide

Minimax problem of GAN

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))]$$

TWO STEP APPROACH

Show that...

1. **The minimax problem of GAN has a global optimum at $p_g = p_{data}$**
2. **The proposed algorithm can find that global optimum**

1. GAN의 minmax problem이 데이터 분포와 G에서 생성된 분포가 동일할때, 글로벌 해를 값을 갖는지 증명
2. 논문에서 제안한 일고리즘(Generative를 뉴얼넷으로 학습시키고, 그다음에 discriminator 학습시키는 반복적인 방법)이 클로벌 해를 찾을 수 있는지 증명이 필요.

THEORETICAL RESULTS

Proposition 1.

For G fixed, the optimal discriminator D is

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}.$$

$$\begin{aligned} C(G) &= \max_D V(G, D) \\ &= \mathbb{E}_{x \sim p_{data}} \left[log D_G^*(x) \right] + \mathbb{E}_{z \sim p_z} \left[log(1 - D_G^*(G(z))) \right] \\ &= \mathbb{E}_{x \sim p_{data}} \left[log D_G^*(x) \right] + \mathbb{E}_{x \sim p_g} \left[log(1 - D_G^*(x)) \right] \\ &= \mathbb{E}_{x \sim p_{data}} \left[log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g} \left[log \frac{p_g(x)}{p_{data}(x) + p_g(x)} \right] \end{aligned}$$

THEORETICAL RESULTS

Proposition 1.

For G fixed, the optimal discriminator D is

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}.$$

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity $V(G, D)$

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

For any $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function $y \to a log(y) + b log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $Supp(p_{data}) \cup Supp(p_g)$, concluding the proof. ■

THEORETICAL RESULTS

Main Theorem

The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-log(4)$.

For $p_g = p_{data}$, $D_G^*(x) = \frac{1}{2}$ and

$$C(G) = \mathbb{E}_{x \sim p_{data}}[-log(2)] + \mathbb{E}_{x \sim p_g}[-log(2)] = -log(4).$$

To show that this is the best possible value of $C(G)$:

$$\begin{aligned} C(G) &= -log(4) + KL\left(p_{data} || \frac{p_{data} + p_g}{2}\right) + KL\left(p_g || \frac{p_{data} + p_g}{2}\right) \\ &= -log(4) + 2 \cdot JSD(p_{data} || p_g). \end{aligned}$$

Here, JSD is always positive value and equal to 0 only if two distributions match.
Therefore, $C^* = -log(4)$ is the global minimum of $C(G)$ where the only solution is $p_g = p_{data}$.

* JSD : https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence

THEORETICAL RESULTS

Convergence of the proposed algorithm

If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{x\sim p_{data}}\left[logD^*_G(x)\right] + \mathbb{E}_{x\sim p_g}\left[log(1-D^*_G(x))\right]$$

then p_g converges to p_{data}.

Proof. Consider $V(G,D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g. The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. This is equivalent to computing a gradient descent update for p_g at the optimal D given the corresponding G, $\sup_D U(p_g, D)$ is convex in p_g with a unique global optima as proven in **Thm 1**, therefore with sufficiently small updates of p_g, p_g converges to p_x, concluding the proof. ■

THEORETICAL RESULTS

Convergence of the proposed algorithm

If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{x\sim p_{data}}\left[logD^*_G(x)\right] + \mathbb{E}_{x\sim p_g}\left[log(1-D^*_G(x))\right]$$

then p_g converges to p_{data}.

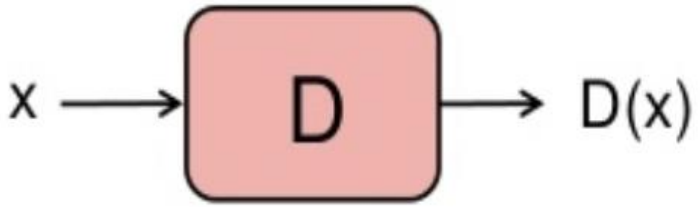
Proof. Consider $V(G,D) = U(p_g, D)$ as a function of p_g as done in the above criterion.

"The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained."

equivalent to computing a gradient descent update for p_g at the optimal D given the

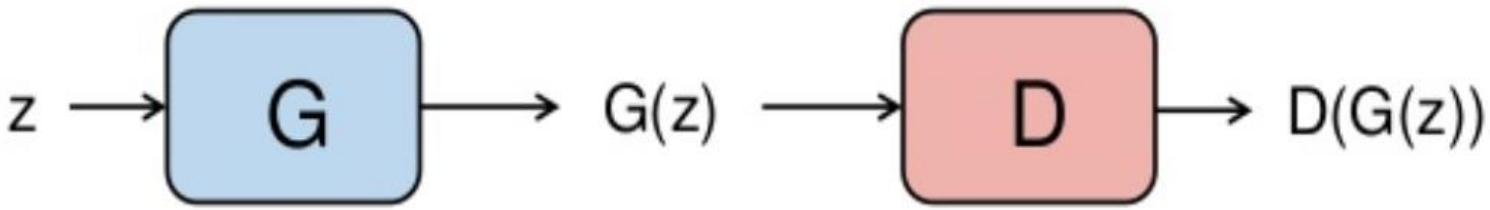
If $f(p_g) = \sup_{D\in\mathcal{D}} f_D(p_g)$ and $f_D(p_g)$ is convex in p_g every D, then $\partial f_{D^*}(p_g) \in \partial f$ if $D^* = arg\sup_{D\in\mathcal{D}} f_D(p_g)$.

- 수식을 잘 이해할 필요가 있음
- 수식이 코드와 일치

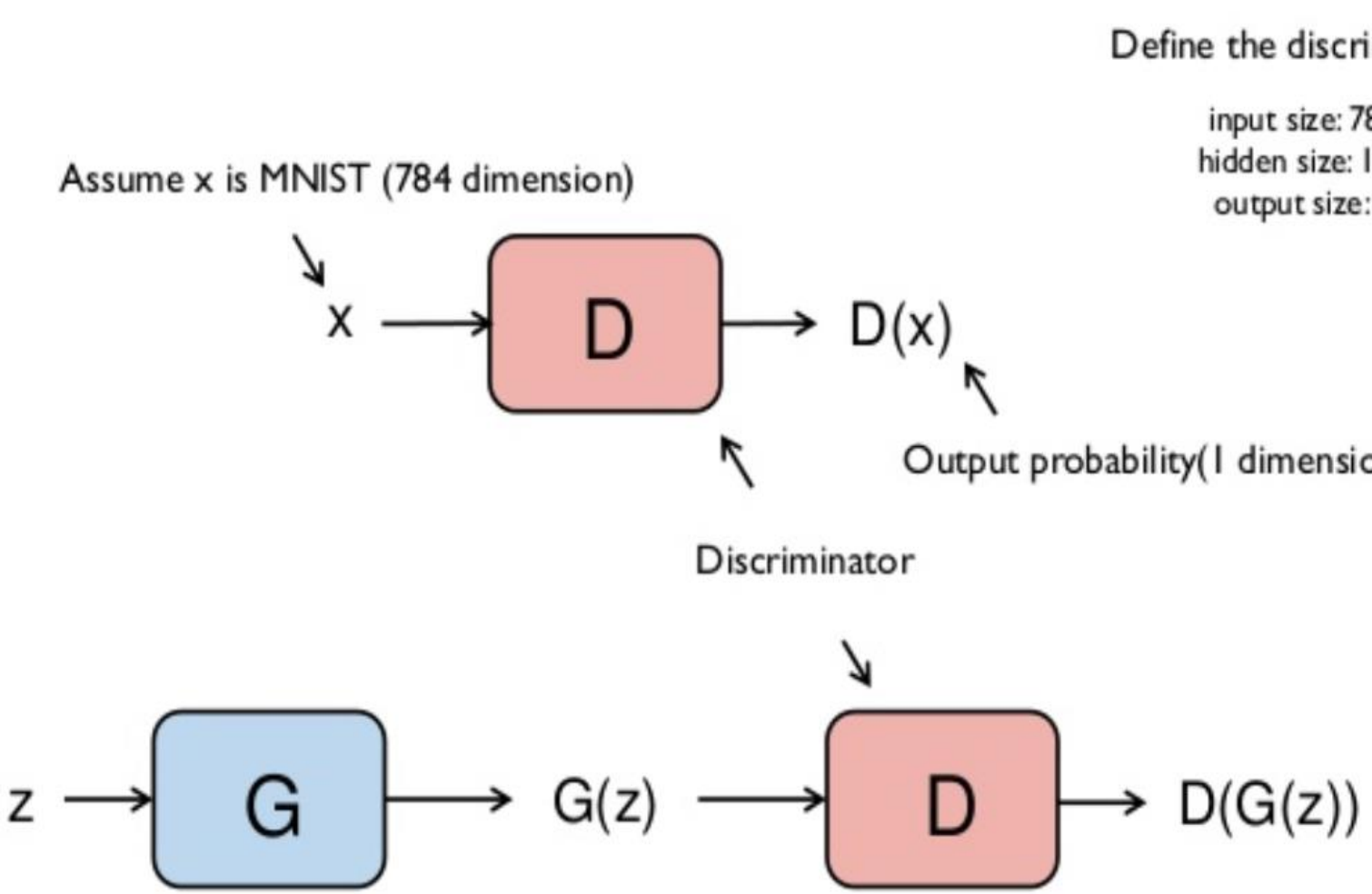


Training with real images

Training with fake images



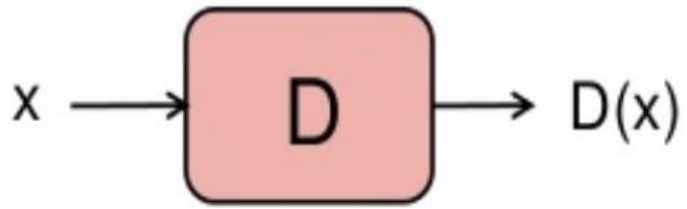

```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```

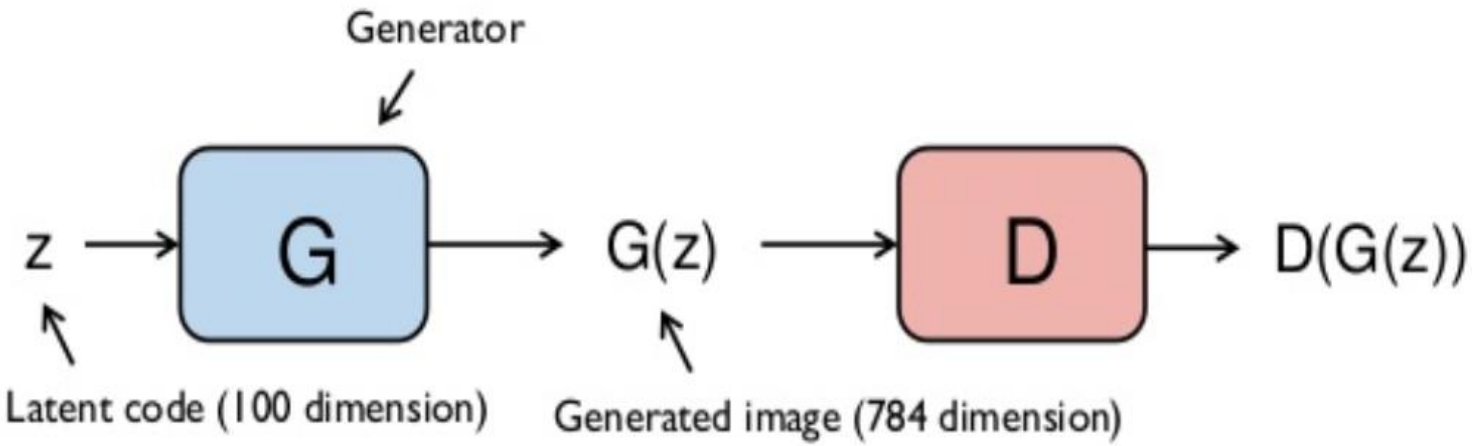


Define the discriminator

input size: 784
hidden size: 128
output size: 1

```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```

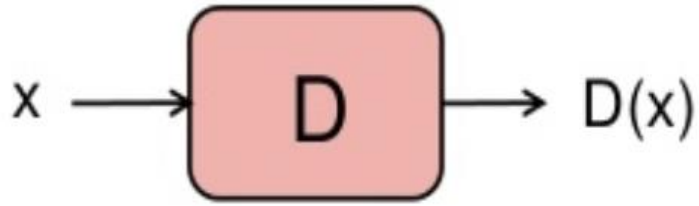




Define the generator

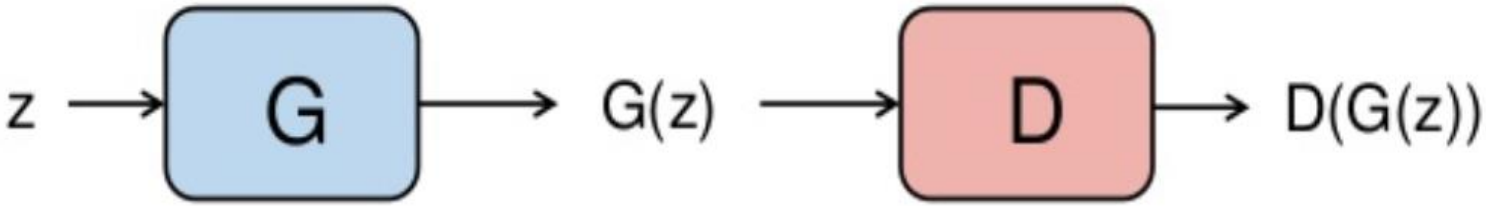
input size: 100
hidden size: 128
output size: 784

```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```

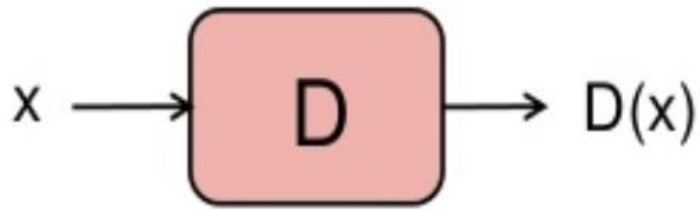


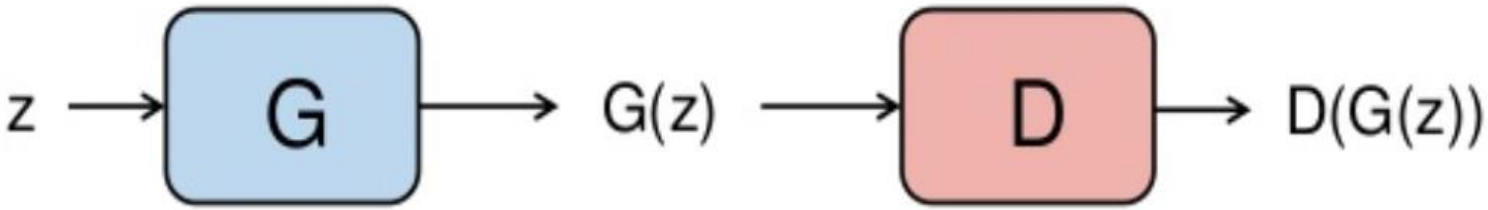
Binary Cross Entropy Loss $(h(x), y)$

$$-y \log h(x) - (1-y) \log(1 - h(x))$$



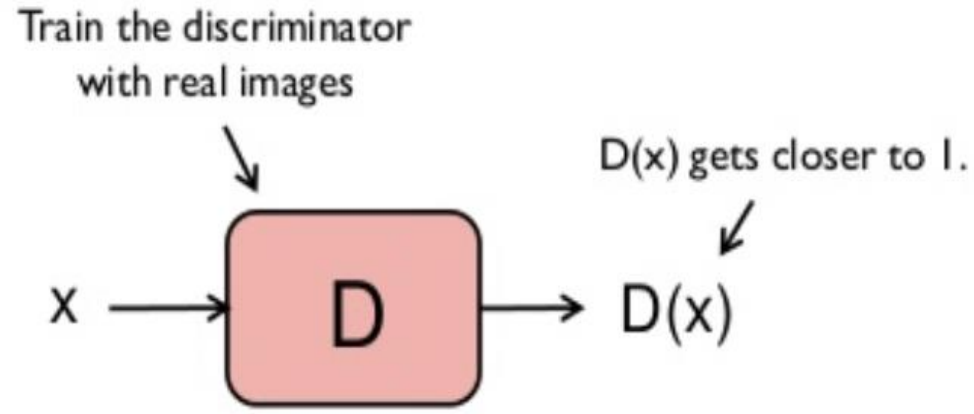

```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```

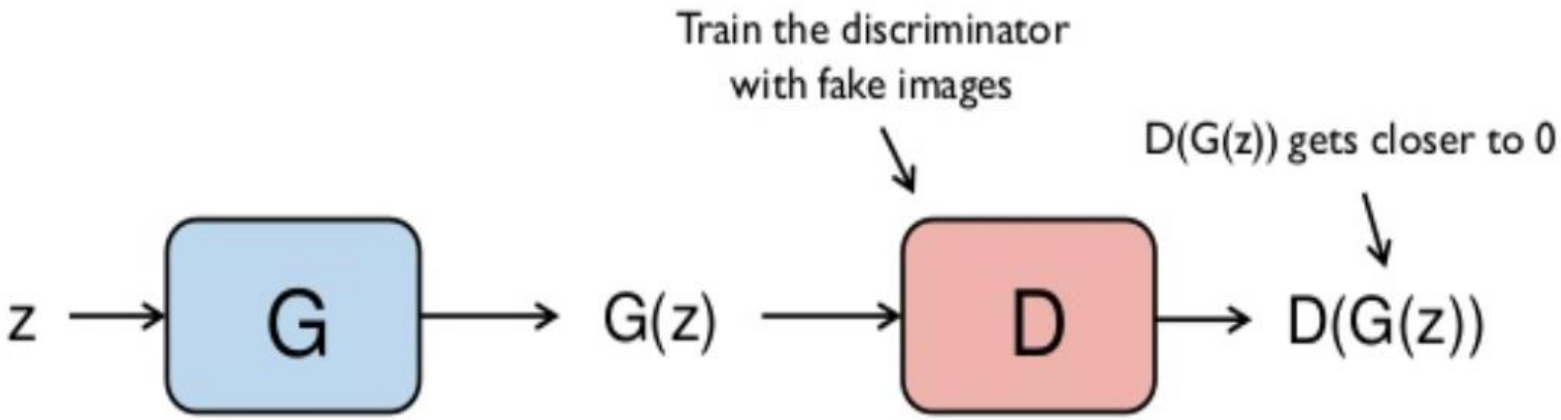




Optimizer for D and G

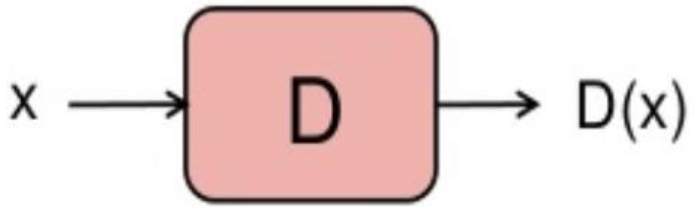
```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```

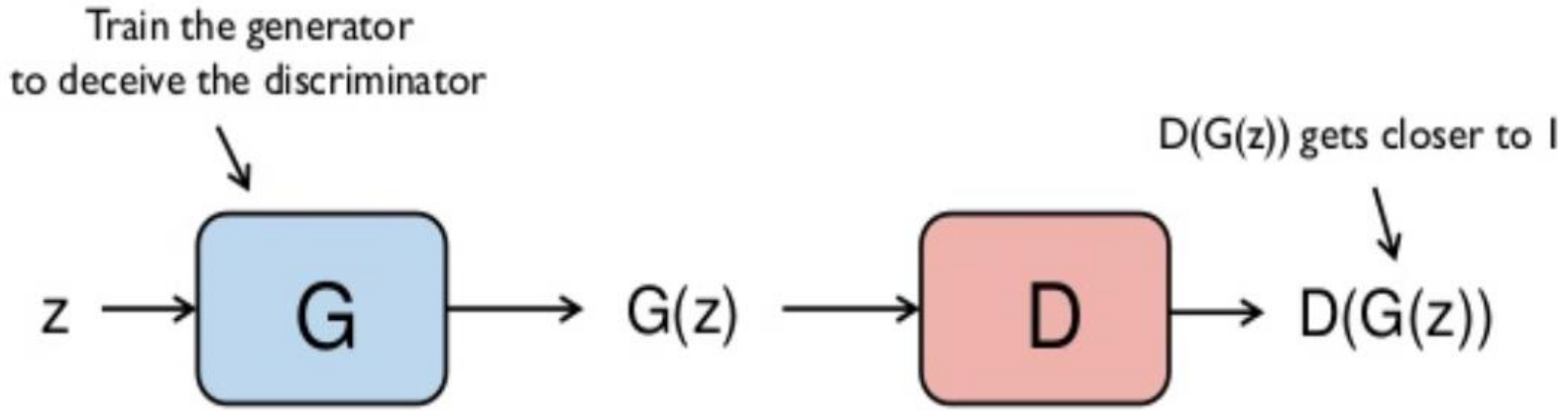




Forward, Backward and Gradient Descent

```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```





https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/02-intermediate/generative_adversarial_network/main.py

```python
import torch
import torch.nn as nn


D = nn.Sequential(
    nn.Linear(784, 128),
    nn.ReLU(),
    nn.Linear(128, 1),
    nn.Sigmoid())

G = nn.Sequential(
    nn.Linear(100, 128),
    nn.ReLU(),
    nn.Linear(128, 784),
    nn.Tanh())

criterion = nn.BCELoss()

d_optimizer = torch.optim.Adam(D.parameters(), lr=0.01)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.01)

# Assume x be real images of shape (batch_size, 784)
# Assume z be random noise of shape (batch_size, 100)

while True:
    # train D
    loss = criterion(D(x), 1) + criterion(D(G(z)), 0)
    loss.backward()
    d_optimizer.step()

    # train G
    loss = criterion(D(G(z)), 1)
    loss.backward()
    g_optimizer.step()
```

Forward, Backward and Gradient Descent →

RESULTS

What can GAN do?

* Figure adopted from DCGAN, Alec Radford et al. 2016 (link)

RESULTS

What can GAN do?

Vector arithmetic
(e.g. word2vec)

$$KING\,(왕) - MAN\,(남자) + WOMAN\,(여자)$$

RESULTS

What can GAN do?

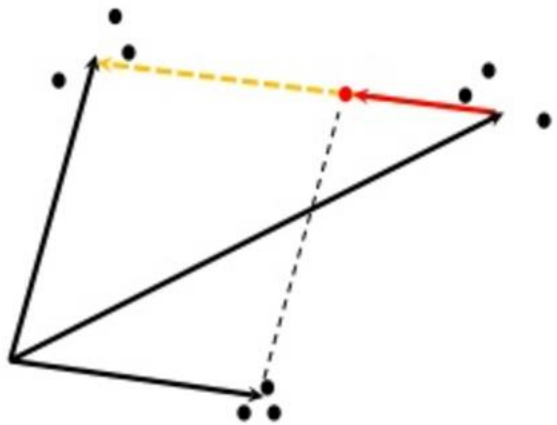
Vector arithmetic
(e.g. word2vec)

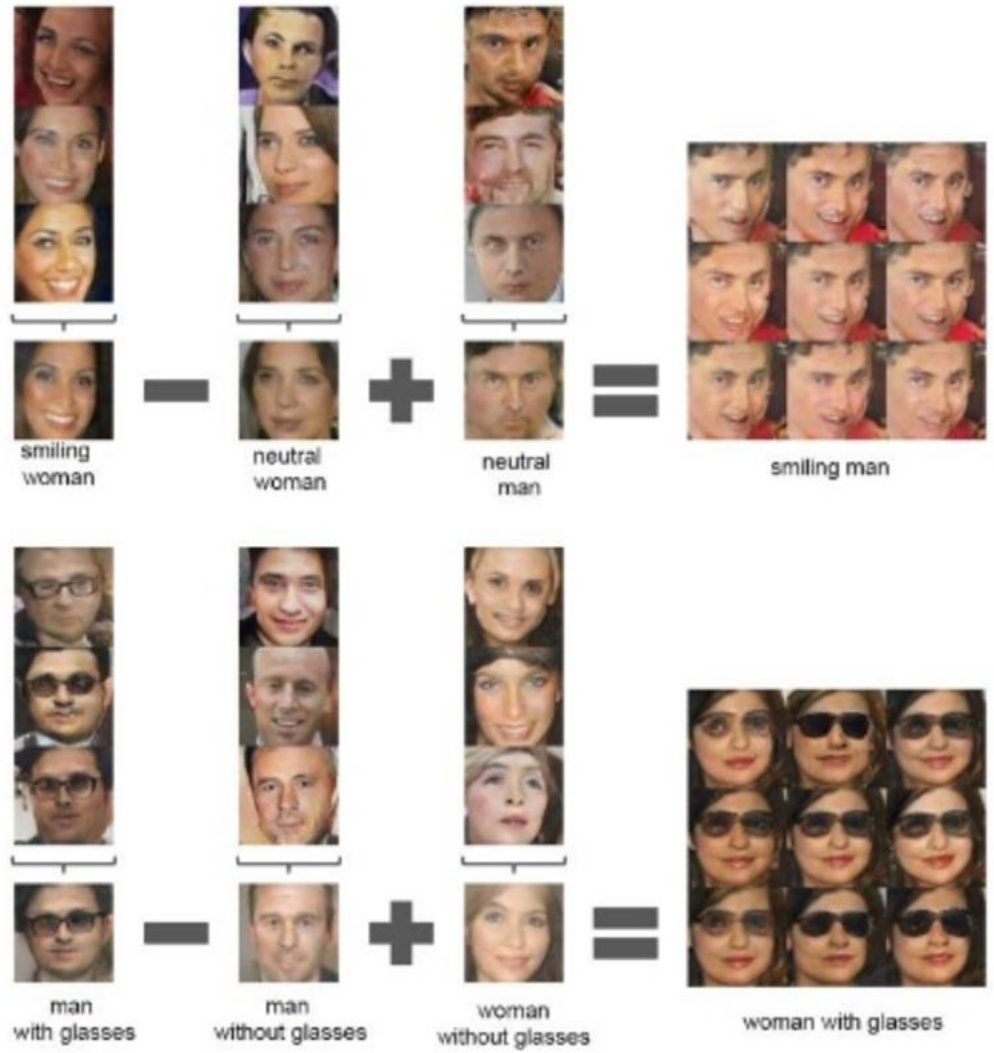
$$QUEEN\ (\text{여왕})$$

RESULTS

What can GAN do?

Vector arithmetic
(e.g. word2vec)





* Figure adopted from DCGAN, Alec Radford et al. 2016 (link)

RESULTS

"We want to get a disentangled representation space EXPLICITLY."

Neural network understanding "Rotation"

* Figure adopted from DCGAN, Alec Radford et al. 2016 (link)

RELATED WORKS

Clip slide

Super-resolution

* SRGAN Christian Ledwig et al. 2017

Img2Img Translation

* CycleGAN Jun-Yan Zhu et al. 2017

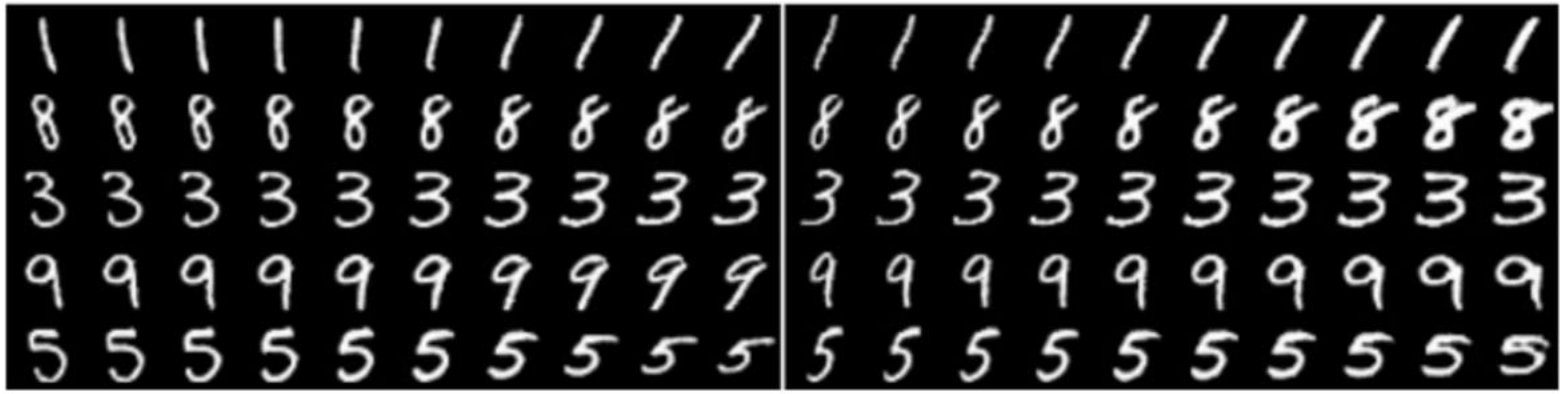
RELATED WORKS

Clip slide

Find a CODE

(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)



(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

* infoGAN Xi Chen et al. 2016

RELATED WORKS

Clip slide

Find a CODE

(a) Rotation

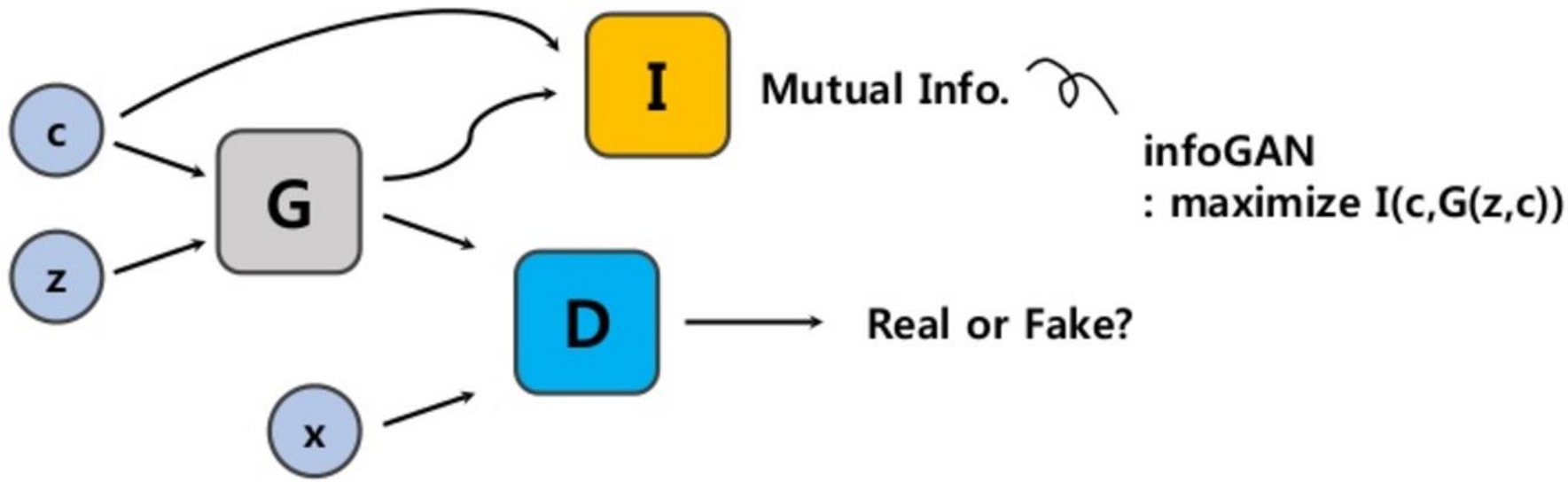
(b) Width

* infoGAN Xi Chen et al. 2016

RELATED WORKS

Diagram of infoGAN

Impose an extra constraint to learn disentangled feature space



"The information in the latent code c should not be lost in the generation process."

THANK YOU ☺

jaejun.yoo@kaist.ac.kr