Generative Models

Part 0,

Introduction

Example 1

자율주행 자동차 / 로봇 시뮬레이션:

트랙 주행은 할 수 있지만, 실제 공도 환경을 재현하기는 어려움.

Example 2

파생상품 시장:

파생상품의 가격 변화를 시뮬레이션해 금융사가 판매할 수 있도록 검증(계리)할 수 있음.

Example 3

이미지 복원/생성:

손상되거나 화질이 낮은 이미지를 복원하거나, 원하는 이미지를 기존 방법보다 훨씬 경제적으로 합성 처리할 수 있음. (딥페이크)

https://qiskit.org/documentation/finance/tutorials/index.html

Part 1,

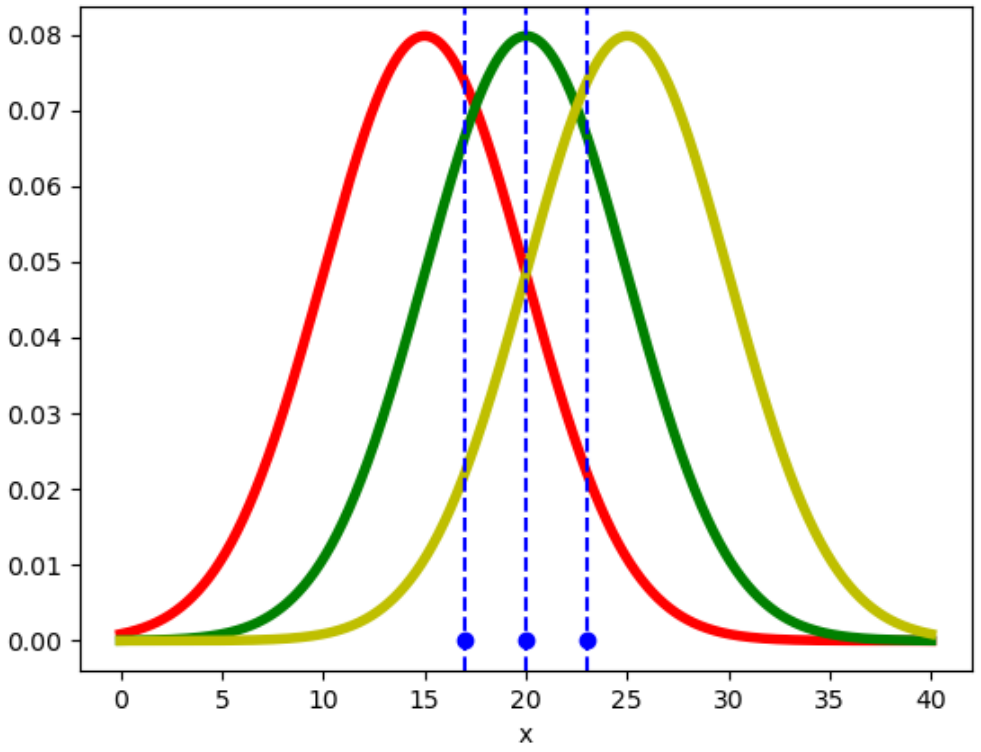
Grounding Knowledge

1. Likelihood, Cross Entropy, KL Divergence
2. Density Estimation
3. Adversarial Game (for GAN)

Part 1,

Likelihood

가능도



A

연속 분포의 그래프에서:
pdf의 값. 특정 사건이 일어날 가능성 ≠ 확률

B

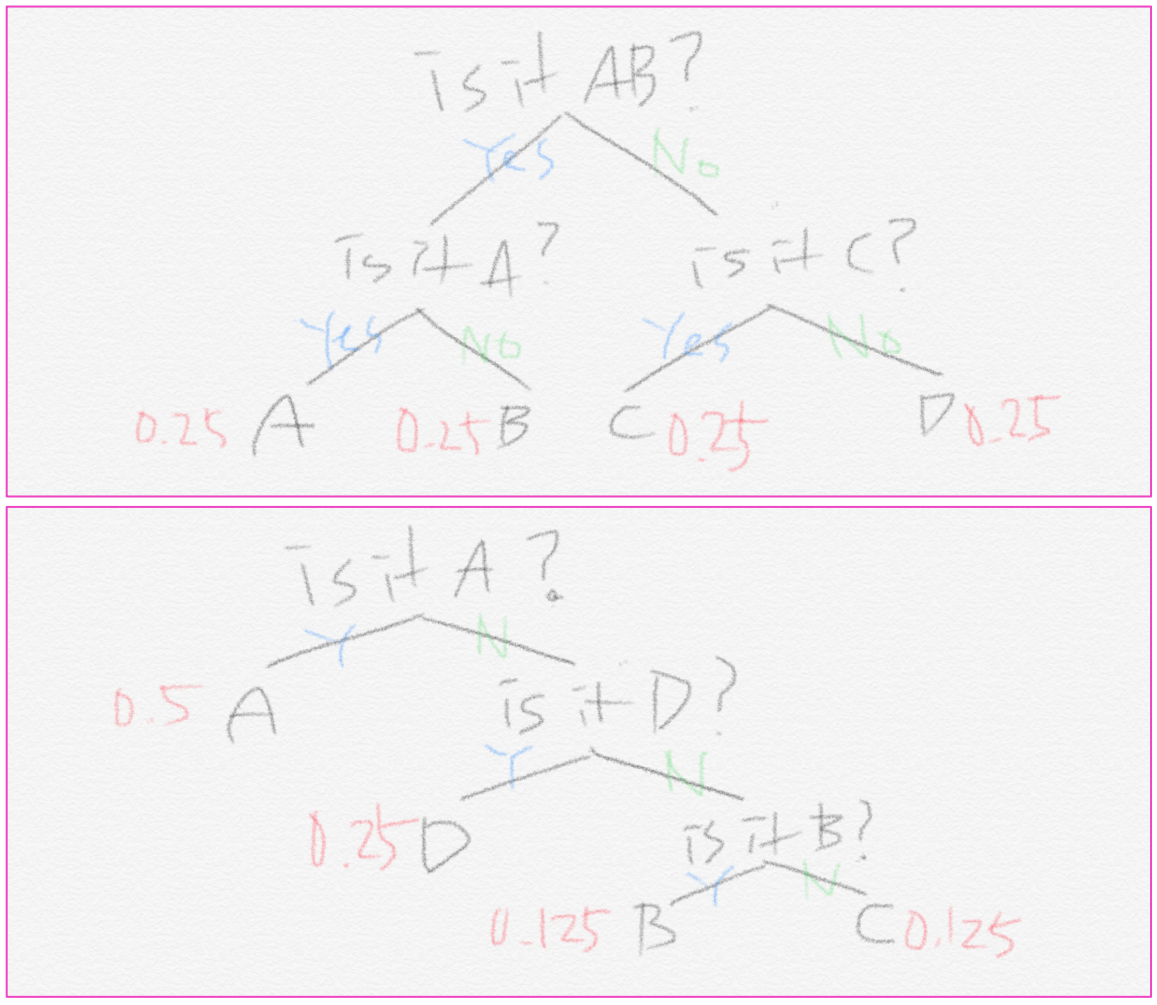
$\prod_i f(x_i;\theta)$:
확률변수의 어떤 파라메터에 대해, 표본 $\{x_i\}$ 가 도출될 가능성.

C

MLE:
likelihood를 최대화하는 $\{x_i\}$를 이용한 통계량. 이때 iid로 추출된 표본일 경우, 계산의 편의성을 위해 log likelihood를 대신 사용해도 무방.

Part 1,

Cross entropy



A

Entropy: 기계 1과 2의 출력 값을 예측할 때, 최적의 전략 하에서 필요한 질문 개수의 기댓값.

$$\sum_{i=1}^{n} p_i \, log_2 \left(\frac{1}{p_i}\right) = -\sum_{i=1}^{n} p_i \, log_2 p_i$$

B

Cross entropy: 기계 2의 출력 값을 예측할 때, 최적의 전략이 아닌 확률 분포 $\{q_i\}$로 표현된 특정 전략을 사용했을 때의 엔트로피 값.

$$H(p, q) = -\sum_{i=1}^{n} p_i \, log_2 q_i$$

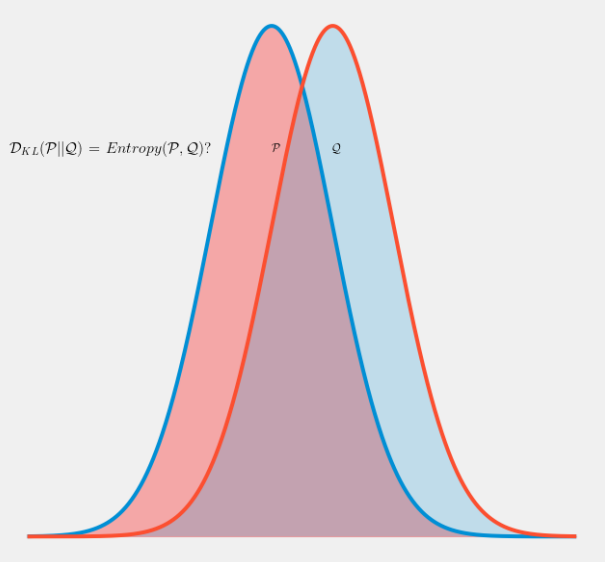
C

Log loss function은 정확히 Cross entropy와 같고, 이것을 최소화하는 것은 negative log likelihood를 최소화하는 문제와 같음.

$$-y log\hat{y} - (1 - y) \log(1 - \hat{y})$$

Part 1,

KL Divergence



$$KL(p \parallel q) = \begin{cases} \sum_i p_i \log \frac{p_i}{q_i} \text{ 또는 } -\sum_i p_i \log \frac{q_i}{p_i} & \text{(이산형)} \\ \int p(x) \log \frac{p(x)}{q(x)} dx \text{ 또는 } -\int p(x) \log \frac{q(x)}{p(x)} dx & \text{(연속형)} \end{cases}$$

A

두 확률 분포의 차이를 계산하는 데에 사용하는 함수. 어떤 이상적인 분포에 대해, 그 분포를 근사하는 다른 분포를 사용해 샘플링을 한다면 발생할 수 있는 정보 엔트로피 차이.

$$KL(p||q) = H(p,q) - H(p)$$

B

1. $KL(p||q) \geq 0$ *by Jensen's inequality*
2. $KL(p||q) \neq KL(q||p)$

* KL divergence는 거리 개념이 아니다.

$$JSD(p||q) = \frac{1}{2}KL(p||M) + \frac{1}{2}KL(q||M)$$

C

$$KL(p||q) \cong \frac{1}{n}\sum_{i=1}^{n}\{-\ln q(x_i;\theta) + \ln p(x_i)\}$$

어떤 분포 p(x)를 q(x)를 이용해 근사시킬 때, KL divergence를 최소화시키는 것은 Likelihood를 최대화하는 것과 같음.

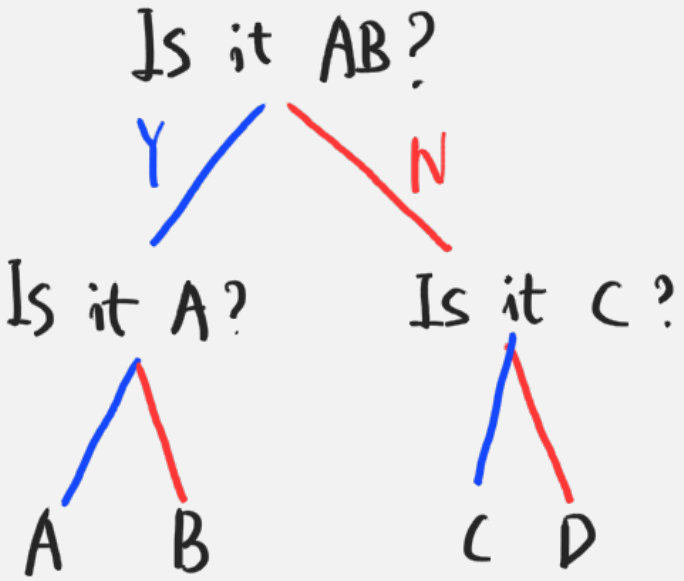
도화지

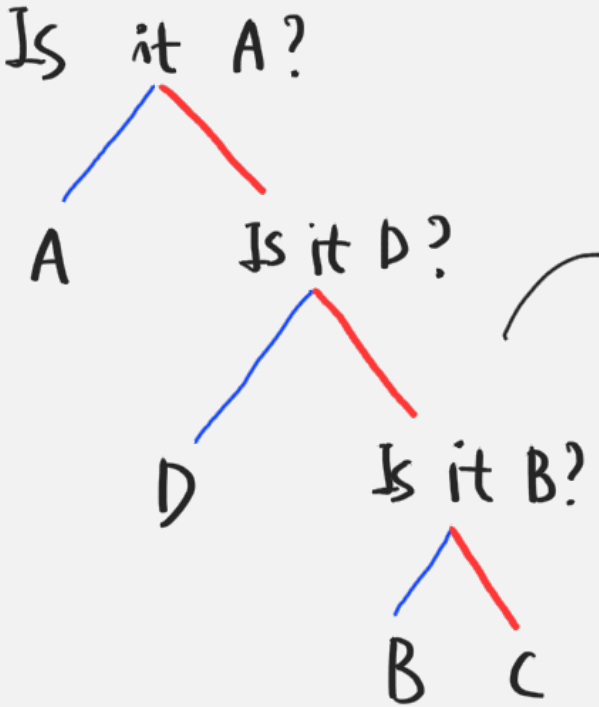
Machine 1

A	B	C	D
1/4	1/4	1/4	1/4

Machine 2

A	B	C	D
1/2	1/8	1/8	1/4





필요한 질문수

: $\log_2$ (가능한 경우의 수)

$= \log_2 \left(\frac{1}{P_i} \right)$

$\log_2(1/1/2)$, $\log_2(1/1/4)$,

$\log_2(1/1/8) = 1, 2, 3$

Entropy H(P) : $\sum P_i \log \left(\frac{1}{P_i} \right)$

$= 4 \cdot 0.25 \cdot 2 = 2$

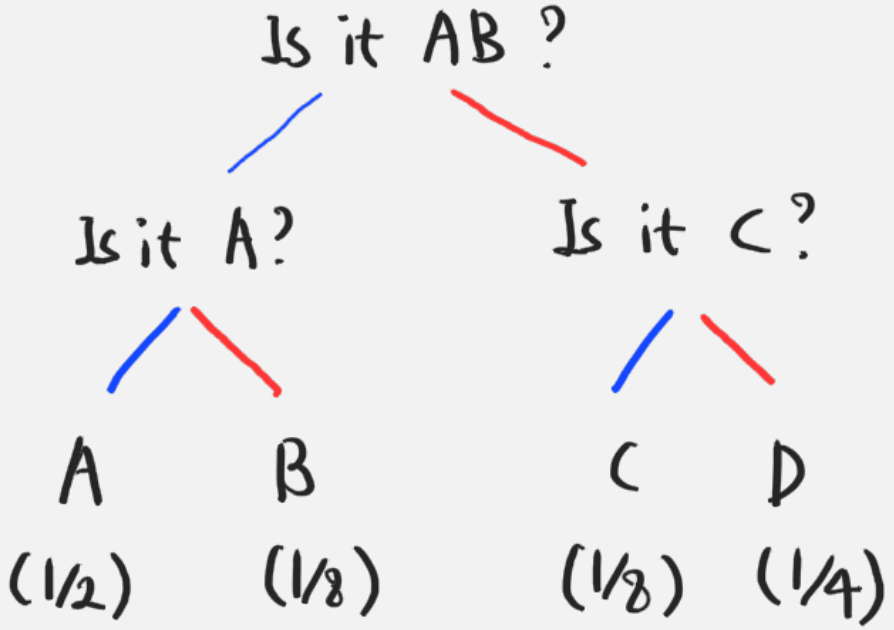
$0.5 \cdot 1 + 0.25 \cdot 2 + 0.125 \cdot 2 \cdot 3$

$= 1.75$

도화지

Machine 2

A	B	C	D
1/2	1/8	1/8	1/4



$P_i = [1/2 \quad 1/8 \quad 1/8 \quad 1/4]$

$q_i = [1/4 \quad 1/4 \quad 1/4 \quad 1/4]$

전략

$\sum p_i \log\left(\frac{1}{q_i}\right) = -\sum p_i \log q_i = 2 = H(P,q)$

전략 q_i가 최적 전략 p_i와 비슷해질수록 작아짐.

≡ Likelihood 최대화 ≡ negative log likelihood 최소화

도화지

$$H(P,Q) = -\sum P_i \log Q_i$$

$$= -\sum P_i \log Q_i + \sum P_i \log P_i - \sum P_i \log P_i$$

$$= \sum P_i \log \frac{P_i}{Q_i} + H(P)$$

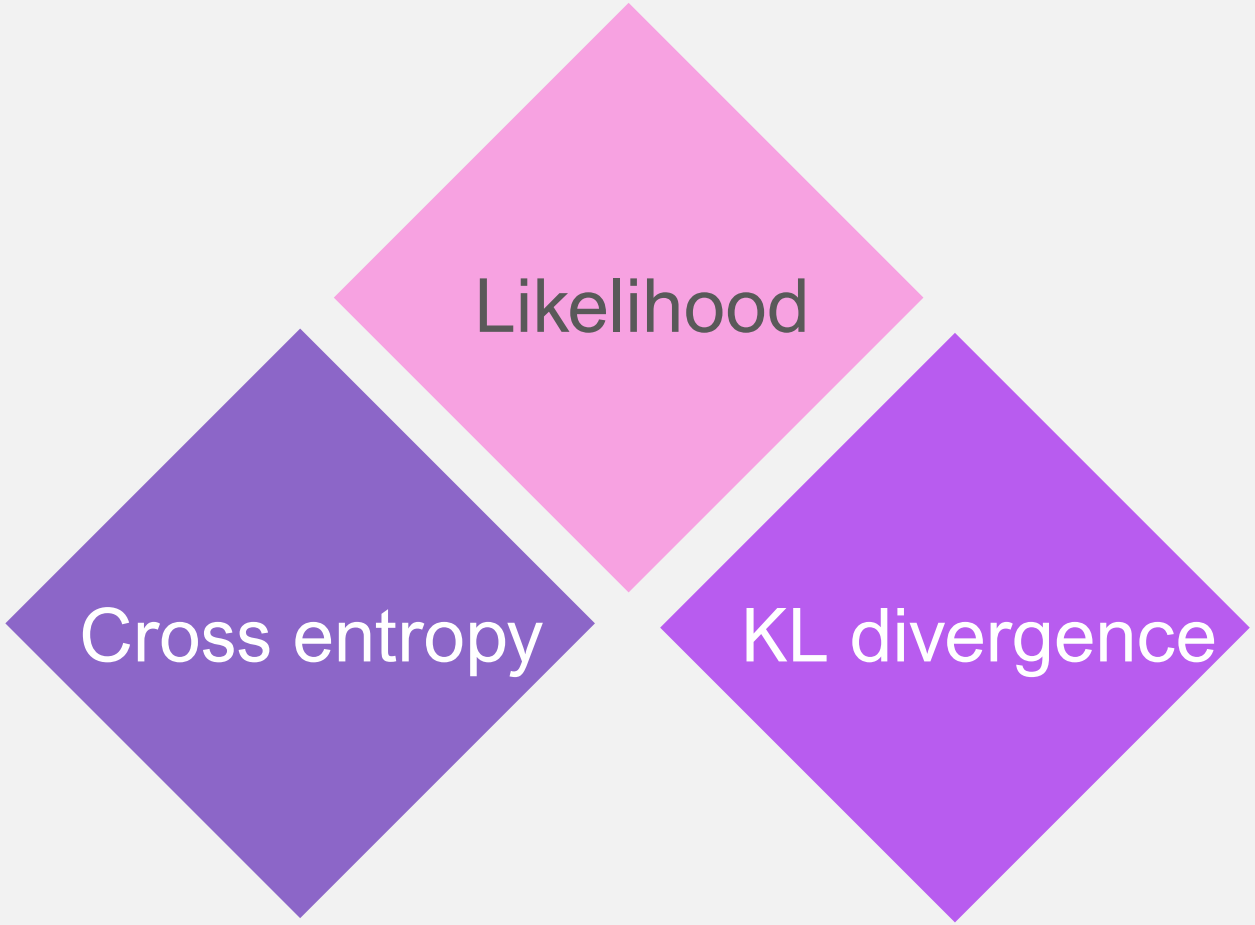
$H(P)$

'정보량 차이' P의 엔트로피

$$KL(P\|Q) = H(P,Q) - H(P)$$

Part 1,

Summary



정보 이론 복습

1. Cross entropy는 negative log likelihood와 같다. 그래서 cross entropy를 최소화하는 것이 log likelihood를 최대화하는 것과 같다.

2. 확률분포 p, q에 대한 cross entropy는 $H(p) + KL(p||q)$와 같으므로 KL divergence를 최소화하는 것 또한 결국 log likelihood를 최대화하는 것과 같다.

Part 2,

Density estimation

분포 추정

A Parametric case

추정하고자 하는 분포 p_{data}와 가장 유사한 p_{model}을 도출할 수 있는 파라메터 θ는?

$$\theta^* = \arg\max_{\theta} E_{x\sim pdata}[\ln p_{model}(x|\theta)]$$

B Non-parametric case

간단한 분포 (uniform, gaussian) 등에 비선형 변환 G를 적용해 p_{data}에 근사한다.

$p_{code}(\mathrm{z})$: $\mathrm{unif}[0,1]$ or $\mathrm{N}(0,1)$ 등. 확률 변수 Z
$\mathcal{Z} = latent\ space$: Z의 확률 공간

$p_{data}(x)$: 근사하고자 하는 분포. 확률 변수 X
$\mathcal{X} = data\ space$: X의 확률 공간

$G: \mathcal{Z} \to \mathcal{X}$ $\mathrm{s.t.}\ \mathrm{G}(\mathrm{z}) = x$: 비선형 변환

Part 2,

Density estimation

분포 추정

A

Single variate case :example

Uniform[0,1]을 코드 분포로 사용한다면?

B

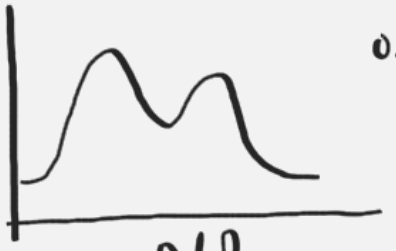
Multivariate case :example

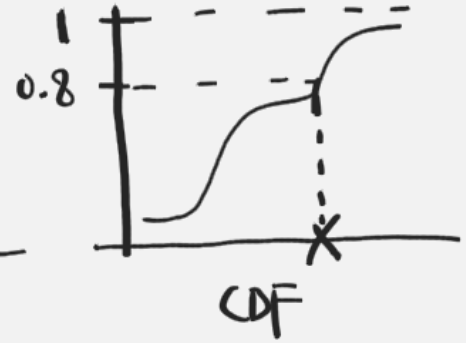
다변량 정규분포 $\mathcal{N}(\mu, \Sigma)$를 근사해 보자.

도화지

P_{code} : unif[0,1] $\mathcal{Z}$: [0,1]

P_{model} , $\mathcal{X}$: 임의의 일변수 확률분포



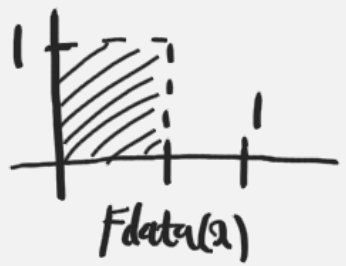


$G: F^{-1}data : [0,1] \rightarrow R$

Assuming $G(Z) = X$, we gain

$$F_{G(Z)}(x) = P(G(Z) \le x) = P(Z \le F_{data}(x)) = F_{data}(x)$$

↑



도화지

$$p_{model}(x) = P_X(x) = \frac{p_{code}(G^{-1}(x))}{|\det J_G(z)|}$$

비선형 변환을 선형 변환으로 근사.

좁은 곳으로 근사 가능.

ex) $p_{model} = \mathcal{N}(\mu, \Sigma) : X$

$p_{code} = \mathcal{N}(0, I_n) : Z$

$G : R^n \to R^n = \mu + Az$, where $\Sigma = AA^T$ Then $G^{-1}(x) = A^{-1}(x-\mu)$, $J = \left|\frac{\partial G}{\partial z}\right| = |\det(A)|$

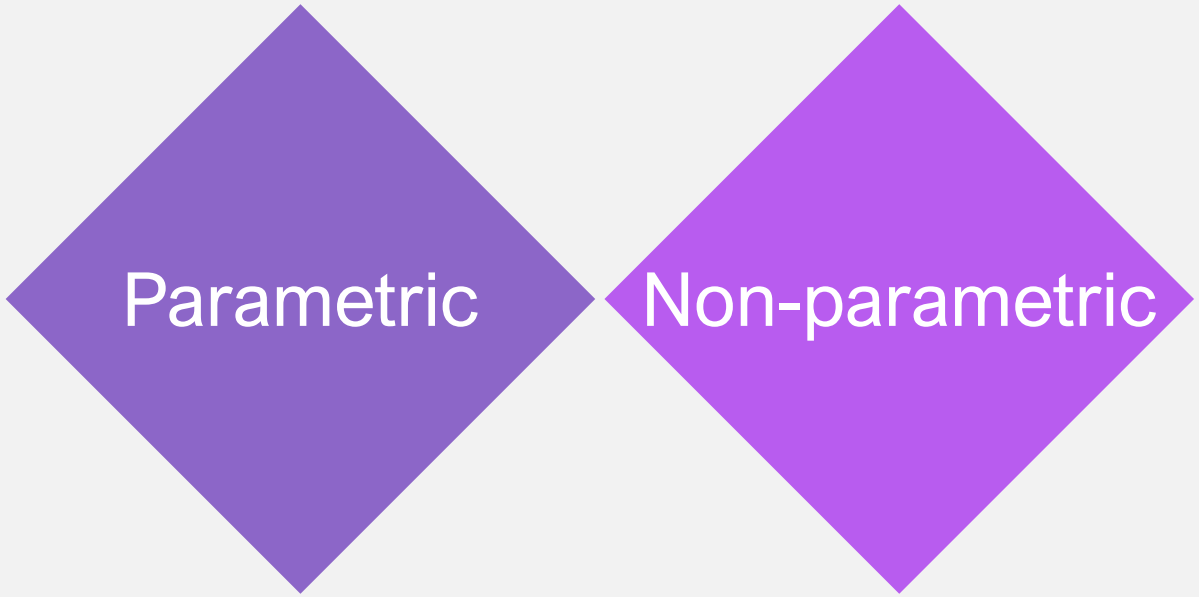
$$p_{code}(z) = \frac{1}{(2\pi)^{2n}} \exp\left(-\frac{1}{2}\|z\|^2\right)$$

$$p_{model}(x) = \frac{1}{|\det(A)|} \frac{1}{(2\pi)^{2n}} \exp\left(-\frac{1}{2}\|A^{-1}(x-\mu)\|^2\right) \Rightarrow \frac{1}{\sqrt{(2\pi)^n \det\Sigma}} \exp\left(-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)\right)$$

교재 595p.

Part 2,

Summary + additional information



Density estimation

1. Parametric case는 MLE 찾기로 해석할 수 있다.

2. Non-parametric case에서 실제로 중요한 문제는 적절한 선형 변환 G를 찾는 문제이다. Latent space의 차원을 최대한 작게 해야 필요한 파라메터의 개수가 줄어들기 때문이다.

3. P model이 parametric/non-parametric 하냐의 여부로 문제를 분류할 수도 있지만, 분포의 형태를 어느 정도 알고 있느냐/알 수 없느냐 여부에 따라서 분류할 수도 있다.

Part 3,

Adversarial games

적대적인 게임

001

두 명의 경쟁자들이, 한 명은 효용 함수를 최소화하고자, 다른 한 명은 최대화하고자 하는 목적을 갖고 각각 parameter 1, parameter 2를 조정.

$$(x^*, y^*) = \arg\max_y \min_x V(x, y)$$

002

Gradient descent로 균형을 찾는다고 할 때, 시간 변수 t에 대해 다음 식으로 나타낼 수 있음.

$$x(t + \Delta t) = x(t) - \Delta t \frac{\partial V}{\partial x}$$

$$y(t + \Delta t) = y(t) + \Delta t \frac{\partial V}{\partial y}$$

003

즉 $\frac{dx}{dt} = -\frac{\partial V}{\partial x}$, $\frac{dy}{dt} = \frac{\partial V}{\partial y}$ 이고, 균형에서 x, y가 변할 유인이 없으므로 도함수 값이 0이어야 하고, 이는 그래디언트 값이 0인 것과 동치.

$$(x^*, y^*) = \lim_{t \to \infty} (x(t), y(t))$$

$$in\ the\ optimal\ point,$$

$$\frac{\partial V}{\partial x} = 0, \qquad \frac{\partial V}{\partial y} = 0$$

도화지

$$V(x,y) = \frac{1}{2}(x^2 - y^2) \qquad \max_y \min_x V(x,y)$$

$$\frac{dx}{dt} = -\frac{\partial V}{\partial x} = -x$$

$$\frac{dy}{dt} = \frac{\partial V}{\partial y} = -y$$

$$\rightsquigarrow \frac{C}{e^t} \quad \Big| \quad \begin{aligned} x(t) &= x_0 e^{-t} \\ y(t) &= y_0 e^{-t} \end{aligned}$$

$$(x^*, y^*) = \lim_{t \to \infty} (x(t), y(t)) = (0,0)$$

discrete case

$$\left.\begin{aligned} x_{n+1} &= x_n - \eta x_n \\ y_{n+1} &= y_n - \eta y_n \end{aligned}\right\} \quad \begin{aligned} x_n &= (1-\eta)^n x_0 \\ y_n &= (1-\eta)^n y_0 \end{aligned} \quad \rightsquigarrow (0,0)$$

Let’s move on!

Variational AutoEncoder

Jeong Yujin

ESC, YONSEI UNIVERSITY

April 6, 2022

Table of Contents

1. Introduction
   - Generative model
   - Latent variable
2. VAE
   - Variational Autoencoder
   - Lower bound
   - Reparameterization trick
   - Loss function
   - code example

I. Introduction

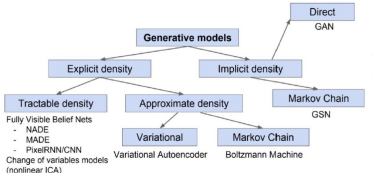
Generative model

Generative model은 주어진 학습데이터를 학습하여 학습 데이터의 분포를 따르는 유사한 데이터를 생성해내는 model이다.

대표적인 model:

- GAN
- VAE

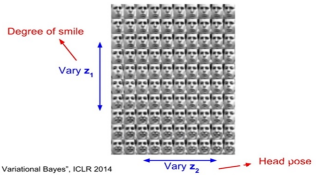
VAE explicitly define and solve for $p_{model}(x)$ and define intractable density function with latent variable **z**.



Latent variable

- Latent variable is a random variable that can not be observed. However, the presence of latent variables can be detected by their effects on variables that are observable.

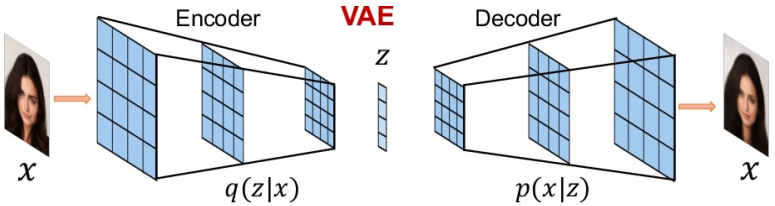
- In VAE, **x** is a data and **z** is latent factors used to generate x.
  e.g.**x** is a face image and **z** can be 'how much of smile is on the face', 'position of the eyebrow', 'orientation of head',...



II. VAE

Variational Auto Encoder

- Variational AutoEncoder는 generative model의 한 종류로, variational bayes와 graphical model에서 영감을 받았다.
- Using **Stochastic Gradient descent method**, VAE learns the parameters ϕ, θ
- VAE approximates **encoder network** $q_\phi(z|x)$ and learns **decoder network** $p_\theta(x|z)$ by using **Neural Network**.
-



VAE

For the case of i.i.d dataset and continuous latent variables per datapoint, the **Auto Encoding VB(AEVB)** algorithm is proposed.

In the AEVB algorithm,

- We make inference and learning efficient by using the SGVB estimator.
- The learned approximate posterior inference model can also be used for recognition, denoising, representation and visualization purposes.
- When a NN is used for the recognition model, we arrive at **variational auto encoder**.

Approximated posterior

- Variational Inference:
  Approximate the taget distribution $p(x)$ by using a tractable density $q(x)$

$$q^* = \underset{q}{argmin} KL(q||p)$$

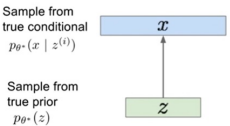
Problem scenario

$\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, i.i.d samples of some continuous and discrete variable $\mathbf{x}$.

We assume that the data are generated by some random process, involving unobserved continuous random variable $\mathbf{z}$.

The random process consists of two steps:

1. a value $\mathbf{z}^{(i)}$ is generated from some prior distribution $p_{\theta^*}(\mathbf{z})$
   $\mathbf{z}^{(i)} \sim p_{\theta^*}(\mathbf{z})$
2. a value $\mathbf{x}^{(i)}$ is generated from some conditional distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$
   $\mathbf{x}^{(i)} \sim p_{\theta^*}(\mathbf{x}|\mathbf{z})$



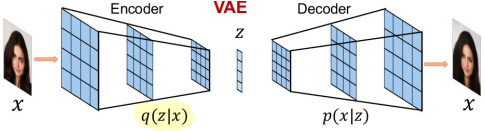
But a lot of this process is hidden from our view: the true parameter θ^* as well as the values of the latent variable $\mathbf{z}^{(i)}$ are unknown to us.

VAE

- In generative model, we want to know $p_\theta(x)$: marginal likelihood.
- However, we can not compute the data likelihood
  $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$
- Posterior density $p_\theta(z|x)$ also intractable because of $p_\theta(x)$

: $p_\theta(z|x) = \frac{p_\theta(x|z)\,p(z)}{p_\theta(x)}$ (by Bayes rule)
(true posterior)

- Then, define additional **encoder network** $q_\phi(z|x)$ that approximates $p_\theta(z|x)$



$q_\phi(z|x)$: **encoder network** $q_\phi(z|x) \sim z$: Gaussian
$p_\theta(x|z)$: **decoder network** $p_\theta(x|z) \sim x$

VAE

- 왜 $q_\phi(z|x)$를 정의했나?
  Because we can derive a **lower bound** on the **data likelihood**.
- $\log p_\theta(x) \geq L(\theta, x)$
  위와 같은 **lower bound(tractable)**를 maximize함으로써 log likelihood를 최대화할 수 있기 때문이다.(maximum likelihood)

VAE

- The marginal likelihood:

$$\log p_\theta(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}) = \sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)})$$

- The marginal likelihoods of individual datapoints can be written as:

$$\begin{aligned}
\log p_\theta(\mathbf{x}^{(i)}) &= E_{z \sim q_\phi(z|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)})\right] \\
&= E_{q_\phi}\left[\log \frac{p_\theta(\mathbf{x}^{(i)}|z)p_\theta(z)}{p_\theta(z|\mathbf{x}^{(i)})}\right] \qquad \left(\because \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)} = \frac{\frac{p_\theta(x,z)}{p_\theta(z)} p_\theta(z)}{\frac{p_\theta(z,x)}{p_\theta(x)}} = p_\theta(x) \right) \\
&= E_{q_\phi}\left[\log \frac{p_\theta(\mathbf{x}^{(i)}|z)p_\theta(z)}{p_\theta(z|\mathbf{x}^{(i)})} * \frac{q_\phi(z|\mathbf{x}^{(i)})}{q_\phi(z|\mathbf{x}^{(i)})}\right]
\end{aligned}$$

VAE

$$= E_{q_\phi}\left[\log \frac{p_\theta(\mathbf{x}^{(i)}|z)p_\theta(z)}{p_\theta(z|\mathbf{x}^{(i)})} * \frac{q_\phi(z|\mathbf{x}^{(i)})}{q_\phi(z|\mathbf{x}^{(i)})}\right]$$

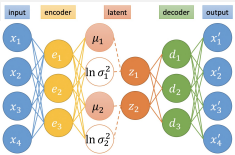
$$= E_{q_\phi(z|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|z)\right] - E_{q_\phi(z|\mathbf{x}^{(i)})}\left[\log \frac{q_\phi(z|\mathbf{x}^{(i)})}{p_\theta(z)}\right] + E_{q_\phi(z|\mathbf{x}^{(i)})}\left[\log \frac{q_\phi(z|\mathbf{x}^{(i)})}{p_\theta(z|\mathbf{x}^{(i)})}\right]$$

$$= E_{q_\phi(z|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|z)\right] - D_{KL}\left(q_\phi(z|\mathbf{x}^{(i)})||p_\theta(z)\right) + D_{KL}\left(q_\phi(z|\mathbf{x}^{(i)})||p_\theta(z|\mathbf{x}^{(i)})\right)$$

① ② ③ ≥ 0 ($p_\theta(z|\mathbf{x}^{(i)})$ is intractable)

- $D_{KL}\left(q_\phi(z|\mathbf{x}^{(i)})||p_\theta(z|\mathbf{x}^{(i)})\right) \geq 0$이므로,
  $L\left(\theta, \phi; \mathbf{x}^{(i)}\right) = E_{q_\phi(z|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|z)\right] - D_{KL}\left(q_\phi(z|\mathbf{x}^{(i)})||p_\theta(z)\right)$: maximize (tractable)
- We can compute estimate of the first term through **sampling**. ($z \sim q_\phi(z|\mathbf{x}^{(i)})$에서 뽑고, $E_{q_\phi}[\log p_\theta(\mathbf{x}^{(i)}|z)]$에 넣어서 계산)
- The second term has nice closed form solution. (if $q_\phi(z|\mathbf{x}^{(i)}) \sim \mathcal{N}(\mu, \text{diag}_\theta(\sigma))$)
- In third term, $p_\theta(z|x)$ is intractable. but ≥ 0

VAE



Hence,

- Variational bound:

$$L\left(\theta, \phi; \mathbf{x}^{(i)}\right) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + E_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

$$\therefore \log p_\theta(\mathbf{x}^{(i)}) \geq L\left(\theta, \phi; \mathbf{x}^{(i)}\right)$$

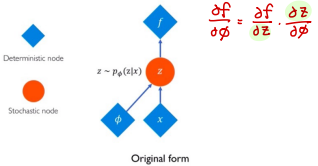
Our goal is to maximize the lower bound $L\left(\theta, \phi; \mathbf{x}^{(i)}\right)$

따라서 lower bound를 최대화하도록 θ와 ϕ를 optimize해야한다.

$$(\theta^*, \phi^*) = \underset{\theta,\phi}{\operatorname{argmax}} L\left(\theta, \phi; \mathbf{x}^{(i)}\right)$$

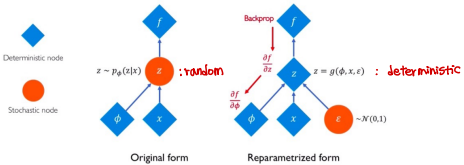
VAE

- How can we optimize ϕ and θ?
  Differentiate and optimize the lower bound w.r.t both the variational parameters ϕ, and generative parameters θ.
- In forward pass, We can compute this term by sampling $z \sim q_\phi(z|x)$. $E_{z\sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right]$.
- 그러나 back propagation은 불가능! sapmling은 미분 가능하지 않다.
  Gradient descent method를 사용하려면 parameter에 대해 미분이 가능해야 하고, model은 deterministic해야한다.(Stochasticity는 input에만 있어야 한다.)



Reparameterization trick

e.g.
t : random
$t = ax+b$: deterministic
↑ random
: 같은 input에 대해선 같은 결과가 나옴.



- 원래 $z \sim q_\phi(z|x)$: 얘는 미분 불가능.
  z : 애 자체가 stochasity.
- 따라서 parameter에 대해 미분 가능하도록 reparameterize해줌.
  $\tilde{z} \sim g_\phi(\epsilon, x)$ with $\epsilon \sim p(\epsilon)$
  : deterministic 해짐 random.
- Reparameterization trick을 사용해서 z를 deterministic 하게 바꿔주었지만, z는 $q_\phi(z|x)$ 에서 sampling 된 것과 마찬가지임.

Take, for example, the univariate Gaussian case: let $z \sim p(z|x) = \mathcal{N}(\mu, \sigma^2)$. In this case, a valid reparameterization is $z = \mu + \sigma\epsilon$, where ϵ is an auxiliary noise variable $\epsilon \sim \mathcal{N}(0, 1)$. Therefore, $\mathbb{E}_{\mathcal{N}(z;\mu,\sigma^2)}[f(z)] = \mathbb{E}_{\mathcal{N}(\epsilon;0,1)}[f(\mu + \sigma\epsilon)] \simeq \frac{1}{L}\sum_{l=1}^{L} f(\mu + \sigma\epsilon^{(l)})$ where $\epsilon^{(l)} \sim \mathcal{N}(0, 1)$.

Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Initialize parameters
repeat
  $\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)
  $\boldsymbol{\epsilon} \leftarrow$ Random samples from noise distribution $p(\boldsymbol{\epsilon})$
  $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \widetilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{X}^M, \boldsymbol{\epsilon})$ (Gradients of minibatch estimator (8))
  $\boldsymbol{\theta}, \boldsymbol{\phi} \leftarrow$ Update parameters using gradients $\mathbf{g}$ (e.g. SGD or Adagrad [DHS10])
until convergence of parameters $(\boldsymbol{\theta}, \boldsymbol{\phi})$
return $\boldsymbol{\theta}, \boldsymbol{\phi}$

VAE

- Now, we can form Monte Carlo estimates of expectations of some function w.r.t $q_\phi(z|x)$ as follows:

$$\underset{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}{E}[f(\mathbf{z})] = \underset{\epsilon\sim p(\epsilon)}{E}\left[f\left(g_\phi(\epsilon, \mathbf{x}^{(i)})\right)\right] \simeq \frac{1}{L}\sum_{l=1}^{L} f\left(g_\phi(\epsilon^{(l)}, \mathbf{x}^{(i)})\right)$$

where $\epsilon^{(l)} \sim p(\epsilon)$

- Apply this technique to the lower bd. $L\left(\theta, \phi; \mathbf{x}^{(i)}\right) = E_{q_{\phi(\mathbf{z}|\mathbf{x}^{(i)})}}[-\log q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) + \log p_\theta(\mathbf{x}^{(i)}, \mathbf{z})]$
- **Stochastic Gradient Variational Bayes** estimator $\tilde{L^A}(\theta, \phi; \mathbf{x}^{(i)}) \simeq L(\theta, \phi, \mathbf{x}^{(i)})$:

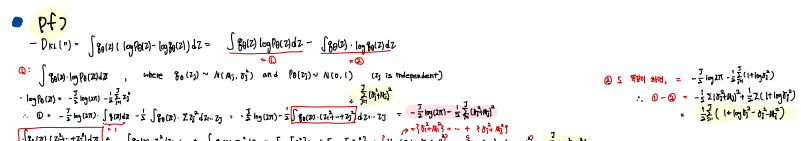
$$\tilde{L^A}(\theta, \phi; \mathbf{x}^{(i)}) = \frac{1}{L}\sum_{l=1}^{L}[\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i,l)}) - \log q_\phi(\mathbf{z}^{(i,l)}|\mathbf{x}^{(i)})]$$

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

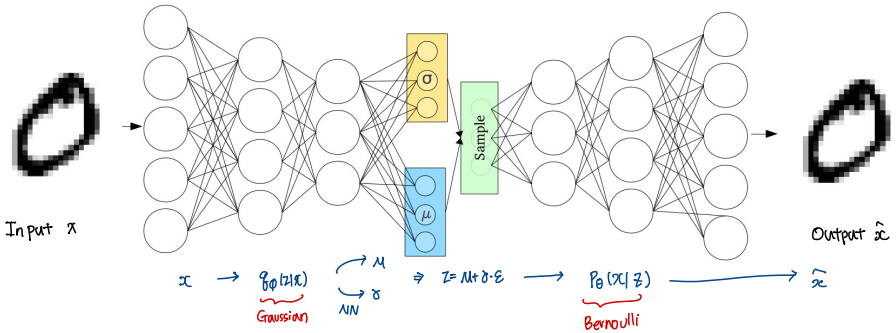
VAE

$$-D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + E_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right]$$

- Often, the KL term $-D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}))$ can be integrated **analytically.** (when $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ is Gaussian)
- only the expected reconstruction error $E_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right]$ requires estimation by **sampling**
- $\tilde{L^B}(\theta, \phi; \mathbf{x}^{(i)}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$ ⇐ $L=1$ in VAE. if minibatch size M is large (eg. $M>100$),
  where $\mathbf{z}^{(i,l)} = g_\phi(\epsilon^{(i,l)}, \mathbf{x}^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$
- $$-D_{KL}((q_\phi(\mathbf{z})||p_\theta(\mathbf{z})) = \int q_\theta(\mathbf{z})\left(\log p_\theta(\mathbf{z}) - \log q_\theta(\mathbf{z})\right) d\mathbf{z}$$
  $$= \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right)$$
- pf)



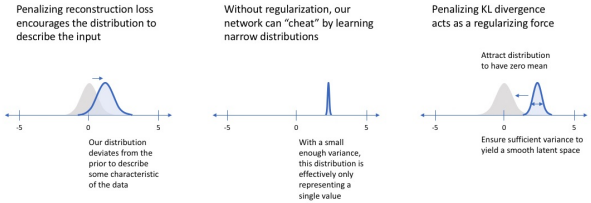
VAE (MNIST)



- $p(z) \sim N(0, I)$
- $q_\phi(z|x) \sim N(\mu_\phi(x), \mathrm{diag}\sigma_\phi(x))$
- $p_\theta(x|z) \sim N(\mu_\theta(z), \sigma_\theta(z))$ or $\mathrm{Bernoulli}(p_\theta(z))$

Loss function

- $-D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}))$: $N(0, I)$
  regularizer역할. $q_\phi(z|x)$가 prior $p_\theta(z)$와 가까워지게함.
  $q(z|x)$가 gaussian distribution을 따르면 analytic하게 계산됨.



- $\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$:
  reconstruction term.
  In our model, $p_\theta(x)$ is multivariate Bernoulli distribution. The $\log p_\theta(x|z)$ is calculated as follows:

In this case let $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ be a multivariate Bernoulli whose probabilities are computed from $\mathbf{z}$ with a fully-connected neural network with a single hidden layer:

$$\log p(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{D} x_i \log y_i + (1 - x_i) \cdot \log(1 - y_i)$$

$$\text{where } \mathbf{y} = f_\sigma(\mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2) \tag{11}$$

where $f_\sigma(.)$ is the elementwise sigmoid activation function, and where $\boldsymbol{\theta} = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\}$ are the weights and biases of the MLP.

- The total loss=reconstruction error(BCE)+KLD

$$\tilde{L^B}(\theta, \phi; \mathbf{x}^{(i)}) = \underbrace{-D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}))}_{\text{minimize}} + \underbrace{\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})}_{\substack{\text{maximize} \\ \Updownarrow \\ \text{minimize binary cross entropy}}}$$

Code Example

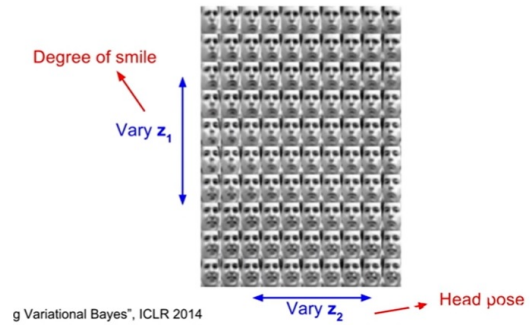
```
# VAE model
class VAE(nn.Module):
    def __init__(self, image_size, hidden_size_1, hidden_size_2, latent_size):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(image_size, hidden_size_1)
        self.fc2 = nn.Linear(hidden_size_1, hidden_size_2)
        self.fc31 = nn.Linear(hidden_size_2, latent_size)
        self.fc32 = nn.Linear(hidden_size_2, latent_size)

        self.fc4 = nn.Linear(latent_size, hidden_size_2)
        self.fc5 = nn.Linear(hidden_size_2, hidden_size_1)
        self.fc6 = nn.Linear(hidden_size_1, image_size)

    def encode(self, x):
        h1 = F.relu(self.fc1(x))
        h2 = F.relu(self.fc2(h1))
        return self.fc31(h2), self.fc32(h2)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + std * eps

    def decode(self, z):
        h3 = F.relu(self.fc4(z))
        h4 = F.relu(self.fc5(h3))
        return torch.sigmoid(self.fc6(h4))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
```

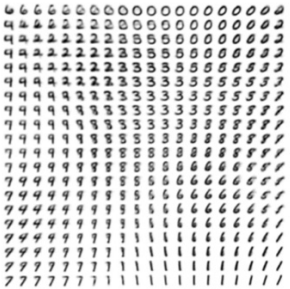
```
def loss_function(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x.view(-1, 784), reduction = 'sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE, KLD
```

```
def train(epoch, model, train_loader, optimizer):
    model.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(DEVICE)
        optimizer.zero_grad()

        recon_batch, mu, logvar = model(data)

        BCE, KLD = loss_function(recon_batch, data, mu, logvar)

        loss = BCE + KLD

        writer.add_scalar("Train/Reconstruction Error", BCE.item(), batch_idx + epoch * (len(train_loader.dataset)/BATCH_SIZE) )
        writer.add_scalar("Train/KL-Divergence", KLD.item(), batch_idx + epoch * (len(train_loader.dataset)/BATCH_SIZE) )
        writer.add_scalar("Train/Total Loss" , loss.item(), batch_idx + epoch * (len(train_loader.dataset)/BATCH_SIZE) )

        loss.backward()

        train_loss += loss.item()

        optimizer.step()

        if batch_idx % 100 == 0:
            print("Train Epoch: {} [{}/{} ({:.0f}%)]\t Loss: {:.6f}".format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader),
                loss.item() / len(data)))

    print("=======> Epoch: {} Average loss: {:.4f}".format(
        epoch, train_loss / len(train_loader.dataset)
    ))
```

VAE

- 따라서 VAE의 장점은 posterior인 $q_\phi(z|x)$와 likelihood인 $p_\theta(x|z)$를 각각 encoder/decoder로 modeling하여 이를 한번에 gradient descent method를 사용해 update한다는 것.
- 단점은 생성된 image들이 다소 blurry하다는 것!





Reference

Auto-Encoding Variational Bayes, Kingma and Maw Welling 2014
https://www.youtube.com/watch?v=5WoItGTWV54
https://cumulu-s.tistory.com/25

GAN
Generative Adversarial Networks

ESC 27기 윤이경

1 Generative Adversarial Networks

- Review : Density Estimation, Adversarial Games
- What is Generative Adversarial Networks?
- GAN _ Goal of Formulation
- Training and Performance Evaluation, Challenges

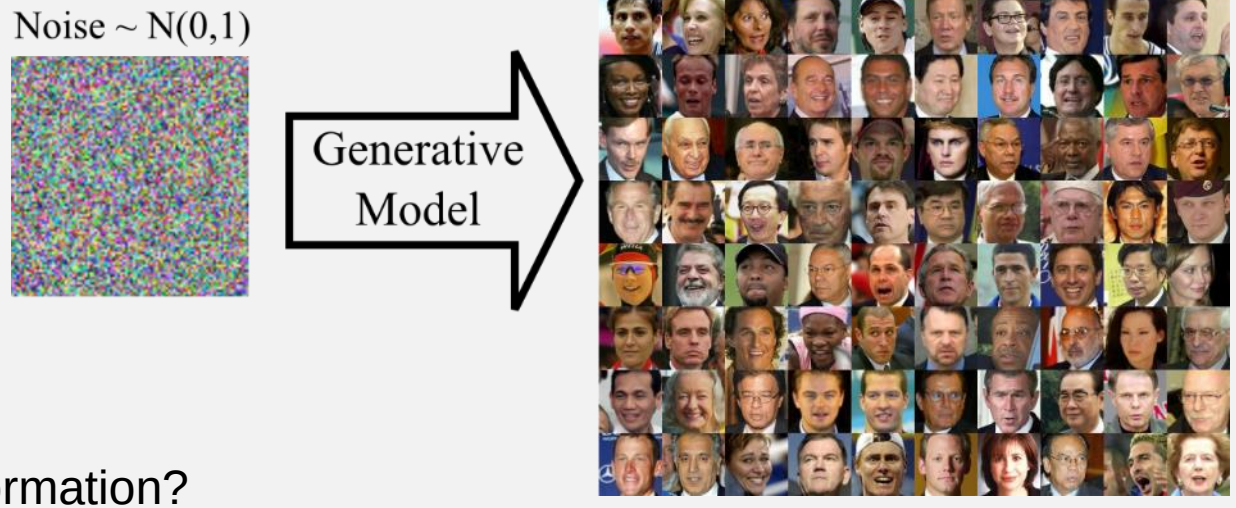
- Generative Moment Matching Networks

Review,

Density Estimation

GAN – GAN just want ability to sample, not explicitly modeling density

1) Sample from a simple distribution (random noise)
2) Learn a transformation to the training distribution.



Q. How to represent this complex transformation?
A. By using a neural net.

Review,

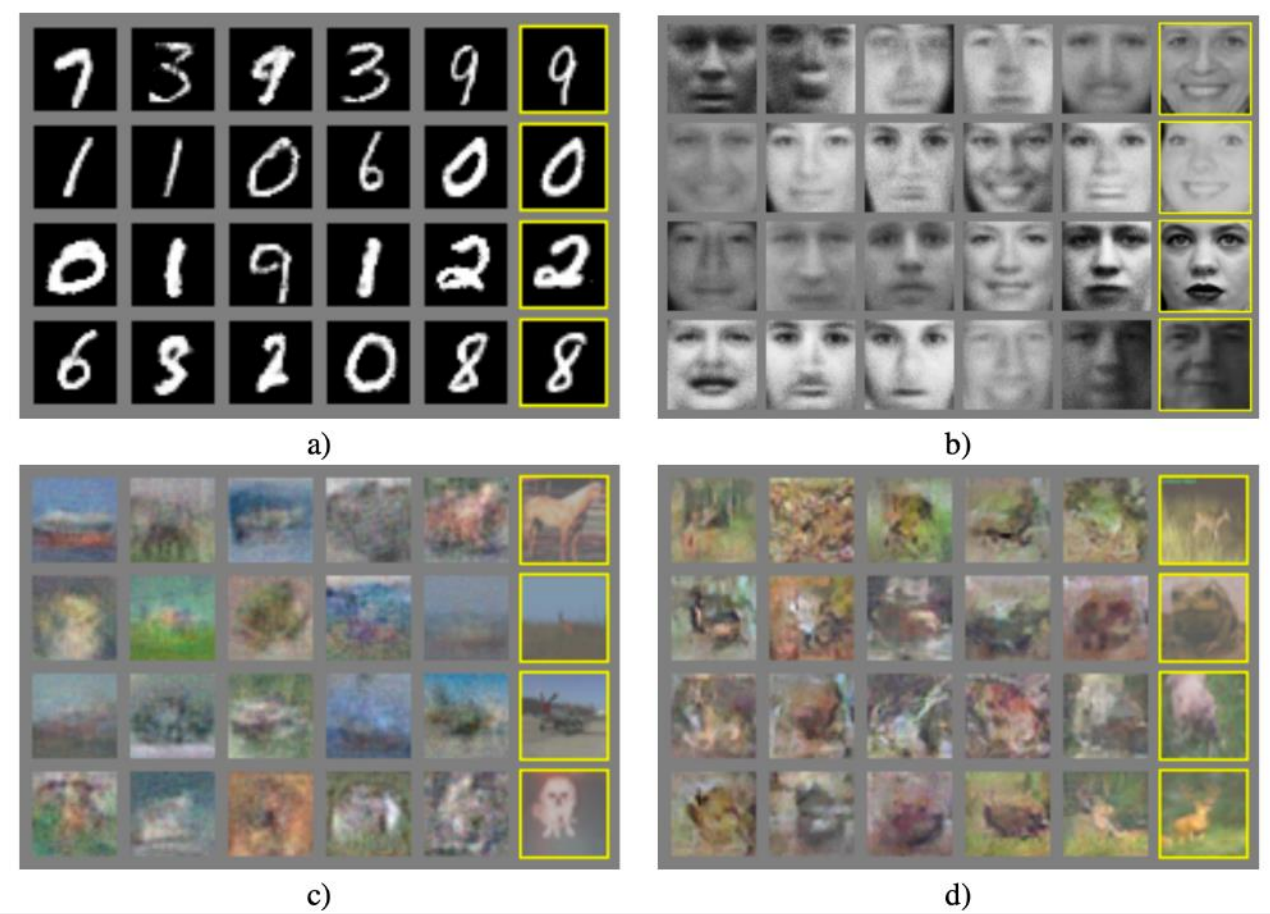
Adversarial Games (Game-theory)

GAN also take a game-theoretic approach.
Learn to generate from training distribution through 2-player game.

$$(x^*, y^*) = \arg\max_y \min_x V(x, y).$$

Part 1,

Generative Adversarial Networks

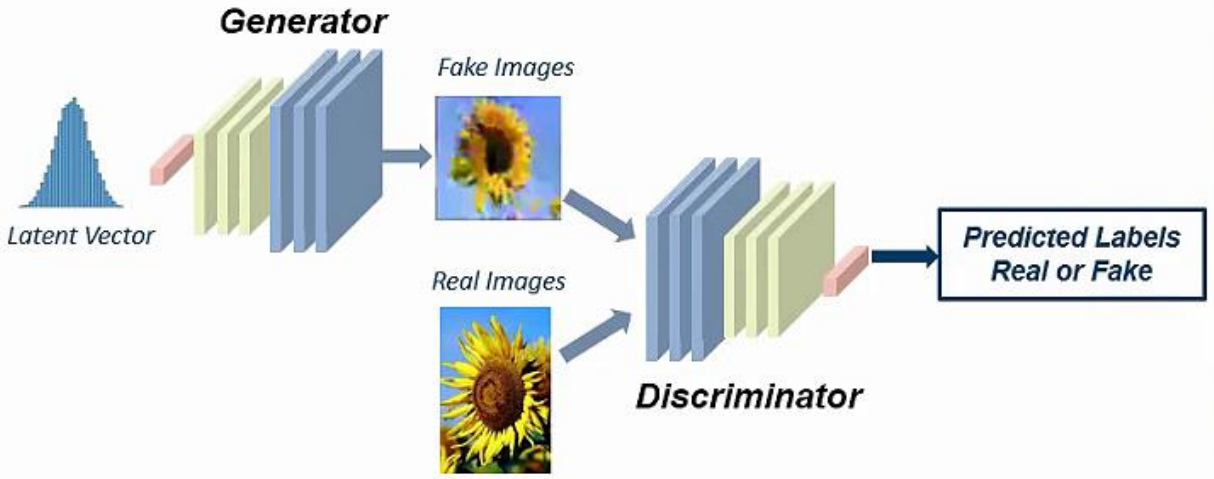


Part 1,

Generative Adversarial Networks

What is GAN(Generative Adversarial Network)?

Train to trick the Discriminator



Train to judge real / fake correctly

Generate an image similar to real images

Part 1,

Generative Adversarial Networks

생성자와 판별자 두개의 네트워크를 활용한 생성 모델.

1) Generator network :

Goal : try to fool discriminator by generating real-looking images

Input : some random noise

Output : image x

$$x = G(z; \theta^{(g)})$$

2) Discriminator network :

Goal : try to distinguish between real and fake images

Input : image x

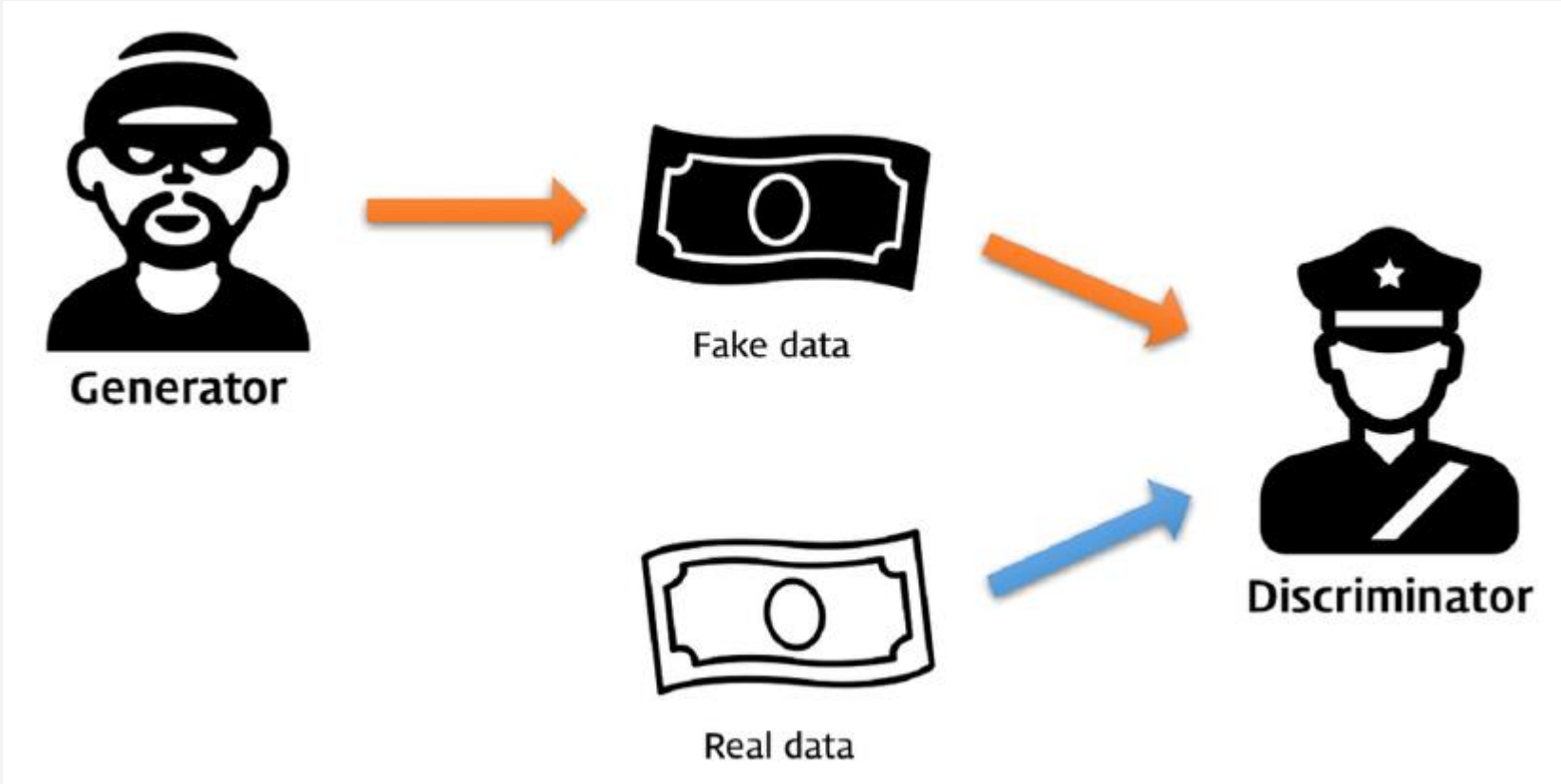
Output : probability between 0 and 1

(real = 1 ~ fake = 0)

$$D(x; \theta^{(d)})$$

Part 1,

Generative Adversarial Networks



Part 1,

Generative Adversarial Networks

VS

판별자 D (Discriminator)

"진짜 화폐랑 G가 만들어내는 가짜 화폐를
잘 구별해야지!!"

생성자 G (Generator)

"진짜 화폐랑 내가 만들어낸 가짜 화폐를
D가 구분할 수 없도록
완벽하게 진짜같은 가짜 화폐를 만들어낼거야!!"

Part 1,

Generative Adversarial Networks

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

$D(x) = 1$일 때 최대

$D(G(z)) = 1$일 때 최소

$D(G(z)) = 0$일 때 최대

Real x

판별자 D
(Discriminator)

D(x)

Training with real image

Training with fake image

Latent code
z

생성자 G
(Generator)

G(z)

판별자 D
(Discriminator)

D(G(z))

Part 1,

Generative Adversarial Networks

Training objectives

$P_{model} \sim G(z)$

Loss function The discriminator tries to maximize the payoff

$$V(G, D) = \mathbb{E}_{x \sim p_{data}}[\ln D(x)] + \mathbb{E}_{x \sim p_{model}}[\ln(1 - D(x))], \qquad (19.4.4)$$

Maximizing the first term assures for a correct classification of true data, while maximizing the second term ensures for the correct classification of the model generated data. This follows from the fact that the discriminator's output $D(x)$ tends to be close to 1 for true data x, while $1 - D(x)$ tends to be close to 1 for generated data x.

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

Generative Adversarial Networks

Minimax problem

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

Discriminator : θ_d wants to maximize objective
- D(x) : how likely x is real → close to 1 (real)
- D(G(z)) : how likely G(z) is fake → close to 0 (fake)

이 두 관계 적대적

Generator : θ_g wants to minimize objective
- D(G(z)) : how likely G(z) is real → close to 1 (look-real)
- discriminator is fooled into thinking that generated G(z) is real

Generative Adversarial Networks

Each gradient update

1. Discriminator update (repeat k times per iteration)
   1) prepare ‘m’ synthetic samples (using generator) and ‘m’ real samples
   2) feed these ‘2m’ samples to discriminator
   3) Gradient ascent : backprop on discriminator network to update θ_d

2. Generator update (once per iteration)
   1) prepare ‘m’ synthetic samples using generator
   2) feed these ‘m’ samples to discriminator
   3) Gradient decent : backprop on whole network
      → backward gradient from discriminator output → generator
      → but update θ_g only

Part 1,

Generative Adversarial Networks

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.

```
data_loader = torch.utils.data.DataLoader(dataset=mnist,
                                          batch_size=batch_size,
                                          shuffle=True)

# Discriminator
D = nn.Sequential(
    nn.Linear(image_size, hidden_size),
    nn.LeakyReLU(0.2),
    nn.Linear(hidden_size, hidden_size),
    nn.LeakyReLU(0.2),
    nn.Linear(hidden_size, 1),
    nn.Sigmoid())

# Generator
G = nn.Sequential(
    nn.Linear(latent_size, hidden_size),
    nn.ReLU(),
    nn.Linear(hidden_size, hidden_size),
    nn.ReLU(),
    nn.Linear(hidden_size, image_size),
    nn.Tanh())

# Device setting
D = D.to(device)
G = G.to(device)

# Binary cross entropy loss and optimizer
criterion = nn.BCELoss()
d_optimizer = torch.optim.Adam(D.parameters(), lr=0.0002)
g_optimizer = torch.optim.Adam(G.parameters(), lr=0.0002)
```

```
def denorm(x):
    out = (x + 1) / 2
    return out.clamp(0, 1)

def reset_grad():
    d_optimizer.zero_grad()
    g_optimizer.zero_grad()
```

https://github.com/yunjey/pytorch-tutorial

```
# Start training
total_step = len(data_loader)
for epoch in range(num_epochs):
    for i, (images, _) in enumerate(data_loader):
        images = images.reshape(batch_size, -1).to(device)

        # Create the labels which are later used as input for the BCE loss
        real_labels = torch.ones(batch_size, 1).to(device)
        fake_labels = torch.zeros(batch_size, 1).to(device)

        # ================================================================== #
        #                      Train the discriminator                       #
        # ================================================================== #

        # Compute BCE_Loss using real images where BCE_Loss(x, y): - y * log(D(x)) - (1-y) * lo
        # Second term of the loss is always zero since real_labels == 1
        outputs = D(images)
        d_loss_real = criterion(outputs, real_labels)
        real_score = outputs

        # Compute BCELoss using fake images
        # First term of the loss is always zero since fake_labels == 0
        z = torch.randn(batch_size, latent_size).to(device)
        fake_images = G(z)
        outputs = D(fake_images)
        d_loss_fake = criterion(outputs, fake_labels)
        fake_score = outputs

        # Backprop and optimize
        d_loss = d_loss_real + d_loss_fake
        reset_grad()
        d_loss.backward()
        d_optimizer.step()
```

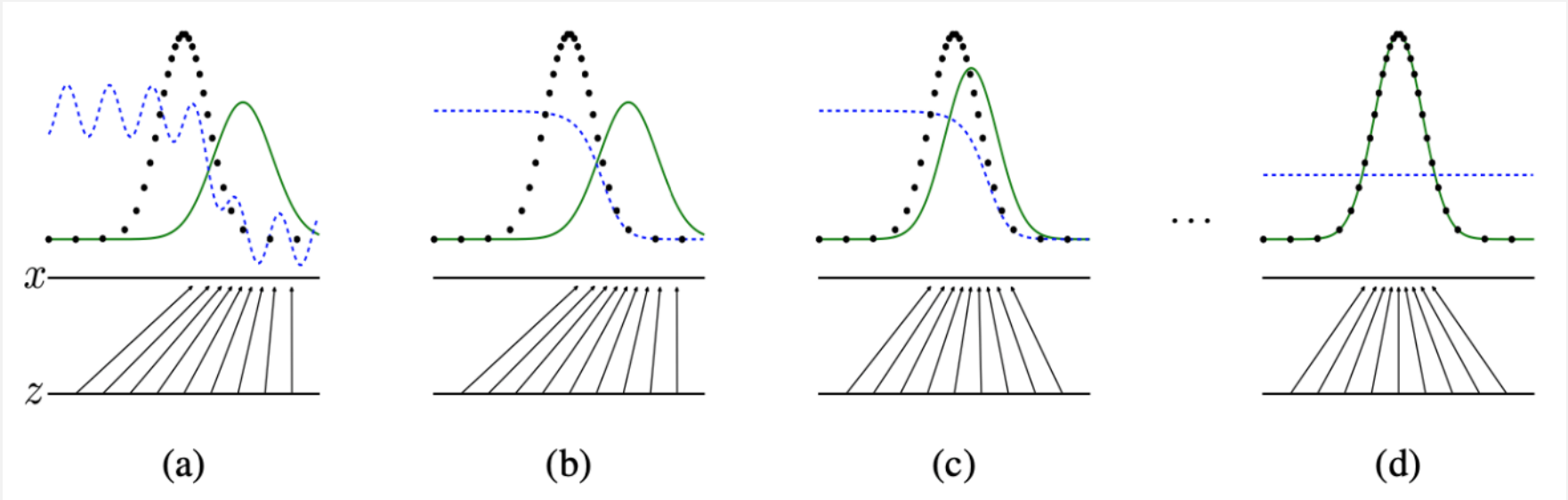
```
        # ================================================================== #
        #                        Train the generator                         #
        # ================================================================== #

        # Compute loss with fake images
        z = torch.randn(batch_size, latent_size).to(device)
        fake_images = G(z)
        outputs = D(fake_images)

        # We train G to maximize log(D(G(z)) instead of minimizing log(1-D(G(z)))
        # For the reason, see the last paragraph of section 3. https://arxiv.org/pdf/1406.2661.pdf
        g_loss = criterion(outputs, real_labels)

        # Backprop and optimize
        reset_grad()
        g_loss.backward()
        g_optimizer.step()

        if (i+1) % 200 == 0:
            print('Epoch [{}/{}], Step [{}/{}], d_loss: {:.4f}, g_loss: {:.4f}, D(x): {:.2f}, D(G(z)): {:.2f}'
                  .format(epoch, num_epochs, i+1, total_step, d_loss.item(), g_loss.item(),
                          real_score.mean().item(), fake_score.mean().item()))

    # Save real images
    if (epoch+1) == 1:
        images = images.reshape(images.size(0), 1, 28, 28)
        save_image(denorm(images), os.path.join(sample_dir, 'real_images.png'))

    # Save sampled images
    fake_images = fake_images.reshape(fake_images.size(0), 1, 28, 28)
    save_image(denorm(fake_images), os.path.join(sample_dir, 'fake_images-{}.png'.format(epoch+1)))

# Save the model checkpoints
torch.save(G.state_dict(), 'G.ckpt')
torch.save(D.state_dict(), 'D.ckpt')
```

Part 1,

Generative Adversarial Networks

Goal of Formulation

$P_{g(z)}(P_{model}) \rightarrow P_{data}$, D(G(z)) = ½ (G(z) is not distinguishable by D)



How can the formulation lead $P_{g(z)}$ converge to P_{data}?

Generative Adversarial Networks

Global optimal

Proposition 19.4.3 *For a fixed generator, G, the optimal discriminator (19.4.5) is given by*

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}. \tag{19.4.6}$$

Part 1,

Generative Adversarial Networks

Proposition: $D_G^*(x) = \dfrac{p_{data}(x)}{p_{data}(x) + p_g(x)}$

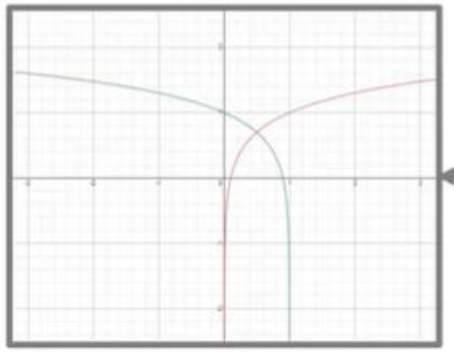
Proof: For G fixed,

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

$$= \int_x p_{data}(x)\log(D(x))\,dx + \int_z p_z(z)\log(1 - D(g(z)))\,dz$$

$$= \int_x p_{data}(x)\log(D(x)) + p_g(x)\log(1 - D(x))\,dx$$

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$



$function\ y \rightarrow alog(y) + blog(1-y)\ achieves\ its\ maximum\ in\ [0,1]\ at \dfrac{a}{a+b}$

same as ***optimal control***: $\dfrac{\delta V(G,D)}{\delta D}[D^*(x)] = 0$

Part 1,

Generative Adversarial Networks

Proof: Considering distributions p_{data} and p_{model} defined on the space $\mathcal{X}$, the payoff function can be written as the following integral:

$$V(G, D) = \int_{\mathcal{X}} \Big[p_{data}(x) \ln D(x) + p_{model}(x) \ln(1 - D(x)) \Big] \, dx.$$

Since G is fixed, we consider the payoff as a functional of $D(x)$ as

$$F(D) = \int_{\mathcal{X}} L(D(x)) \, dx,$$

with the Lagrangian function

$$L(G) = p_{data}(x) \ln D(x) + p_{model}(x) \ln(1 - D(x)).$$

Critical points satisfy the variational equation

$$\frac{dL(D)}{dD} = 0.$$

This becomes

$$\frac{p_{data}(x)}{D(x)} - \frac{p_{model}(x)}{1 - D(x)} = 0.$$

Solving for D we obtain the solution

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}.$$

In order to show that this critical point corresponds to a maximum, we consider the second variation and show that it has a negative value

$$\frac{d^2 L(D)}{dD^2} = -\frac{p_{data}(x)}{D(x)^2} - \frac{p_{model}(x)}{(1 - D(x))^2}.$$

Hence, $D^*_G(x)$ corresponds to a maximum of $V(G, D)$ for a given G. ■

Generative Adversarial Networks

Global optimum point

Proposition 19.4.6 *The global minimum*

$$G^* = \arg\min_G V(G, D_G^*)$$

is achieved if and only if $p_{model} = p_{data}$. *At this minimum point the value is* $-\ln 4$.

Part 1,

Generative Adversarial Networks

Proposition: *Global optimum point is* $\boldsymbol{p_g = p_{data}}$

Proof:

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

$$= E_{x\sim p_{data}(x)}\left[log\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] + E_{x\sim p_g(x)}\left[log\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right]$$

$$= E_{x\sim p_{data}(x)}\left[log\frac{2 * p_{data}(x)}{p_{data}(x)+p_g(x)}\right] + E_{x\sim p_g(x)}\left[log\frac{2 * p_g(x)}{p_{data}(x)+p_g(x)}\right] - \log(4)$$

$$= KL(p_{data}||\frac{p_{data}(x)+p_g(x)}{2})) + KL(p_g||\frac{p_{data}(x)+p_g(x)}{2}) - \log(4)$$

$$= 2 * JSD(p_{data}||p_g) - \log(4)$$

removed when $p_g = p_{data}$

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

$$KL(p_{data}||p_g) = \int_{-\infty}^{\infty} p_{data}(x)\log\left(\frac{p_{data}(x)}{p_g(x)}\right)dx$$

$$JSD(p||q) = \frac{1}{2}KL(p||\frac{p+q}{2}) + \frac{1}{2}KL(q||\frac{p+q}{2})$$

- The *Kullback-Leibler* (KL) divergence

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log \left(\frac{P_r(x)}{P_g(x)} \right) P_r(x) d\mu(x) \ ,$$

- The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m) \ ,$$

Generative Adversarial Networks

Performance evaluation

Desideratum #1 : Quality

- low entropy (highly predictable) conditional label distribution $p(y|x)$
  = given an image x, should know its label y easily
- use neural net to classify generated images → gives distribution $p(y|x)$

Desideratum #2 : Diversity

- high entropy (less predictable) $p(y)$

$$p(y) = \int p\,(\,\mathrm{y}\mid \mathrm{x} = \mathrm{G}(\mathrm{z})\,)\mathrm{dz}$$

= if generated images are diverse, p(y) should be uniform

There is often trade-off between quality and diversity

Part 1,

Generative Adversarial Networks

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Performance evaluation

1. IS: Inception Score

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x} \sim p_g} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right),$$

2. FID: Frechet Inception Distance

$$FID = \|\mu_r - \mu_g\|^2 + T_r(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

= actually compares statistics of generated samples to real sample

Part 1,

Generative Adversarial Networks

GAN Training challenges

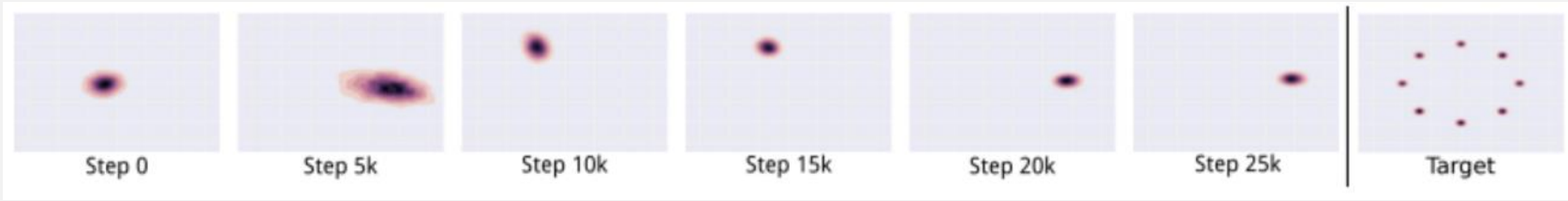
1. non-convergence : Model parameters oscillate/ destabilize/ never convergence.

Problem in GAN training :
- downhill move of one player -> may push the other player uphill
- a general problem with games not unique to GAN

In theory : updates in function space : guaranteed convergence (a convex problem)

In practice : updates in parameter space (non-convex) : no such guarantee

Consequences :
- oscillation :
- mode collapse :



Part 1,

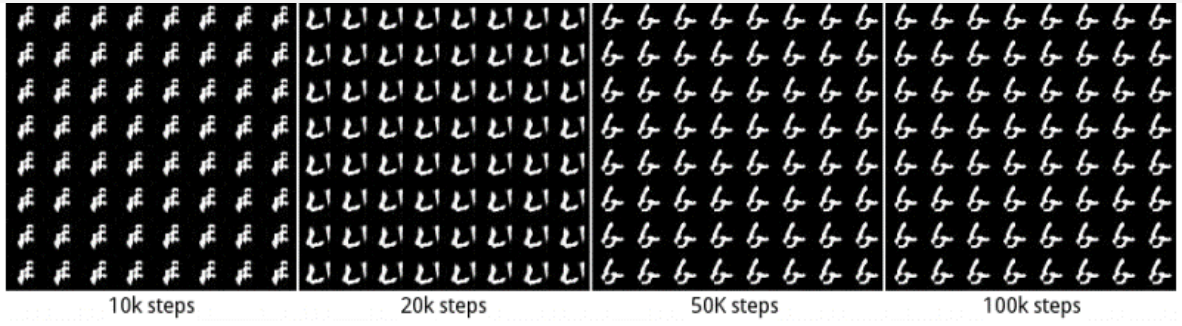
Generative Adversarial Networks

GAN Training challenges

2. Mode collapse : most common form of harmful non-convergence
: several different input z values to the same output point

: Discriminator와 generator가 학습이 서로 상호작용을 하면서 같이 학습이 진행 되어야 하는데,
한쪽이 너무 학습이 잘 되어버려서(학습의 불균형) mode collapse가 발생

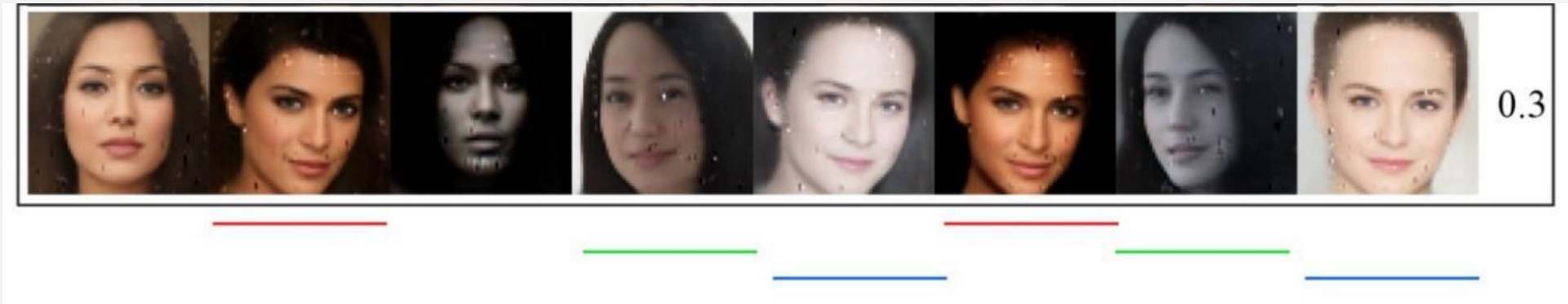
: Discriminator와 generator 사이의 능력에 적절한 균형을 이루면서,
두 네트워크가 안정적으로 전역해로 수렴하도록 만드는 것이 GAN이 해결해야 할 숙제



Part 1,

Generative Adversarial Networks

Partial mode collapse

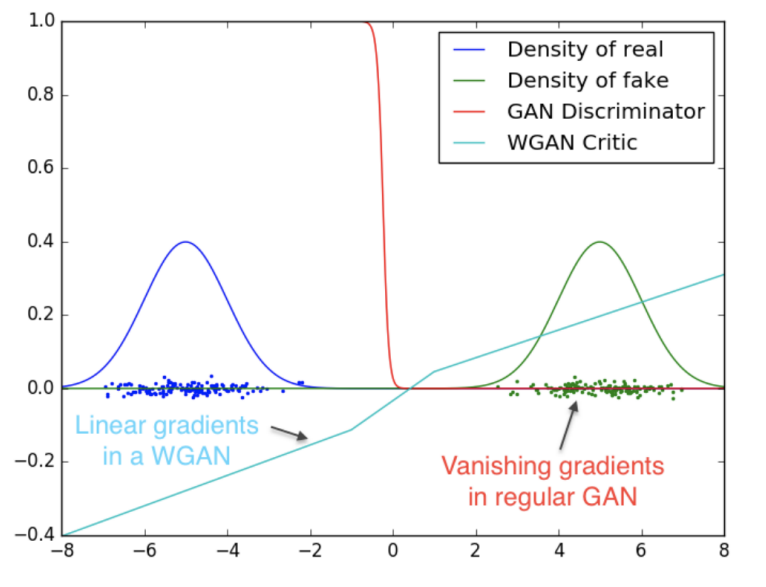


Part 1,

Generative Adversarial Networks

딥러닝 GAN 튜토리얼 - 시작부터 최신 트렌드까지 GAN 논문 순서 | mocha's machine learning (ysbsb.github.io)

: Wasserstein GAN
: Deep Convolutional GAN (DCGAN)
: Least Squares GAN (LSGAN)
: Semi-Supervised GAN (SGAN)
: Auxiliary Classifier GAN (ACGAN)
: CycleGAN



name change : discriminator → critic
√ GAN discriminator : predict the probability of generated images being real
√ WGAN critic : score the realness/fakeness a given image

Part 1,

Generative Moment Matching Networks

Generative moment matching networks have been introduced by Li et al in 2015.

The generator is trained by ‘moment matching’, a technique by which the generator’s outcome has moments as close as possible to the corresponding moments of training data.

⇒ consider the parameter θ that minimizes the maximum mean discrepancy defined by

$$\theta^* = \arg\min_\theta d_{MMD}(p_\theta, q) = \arg\min_\theta \|\mu_\phi(G(Z;\theta)) - \mu_\phi(Y)\|_{Eu}.$$

$$\theta^* = \arg\min_\theta \left\{ \frac{1}{n^2} \sum_{i,j} K(x_i, x_j) + \frac{1}{m^2} \sum_{i,j} K(y_i, y_j) - \frac{2}{mn} \sum_{i,j} K(x_i, y_j) \right\},$$

* maximum mean discrepancy(MMD) : distance between two distributions