[12주차]
Generative Models

1기 강다연
1기 김연수
1기 김지인
1기 장예서

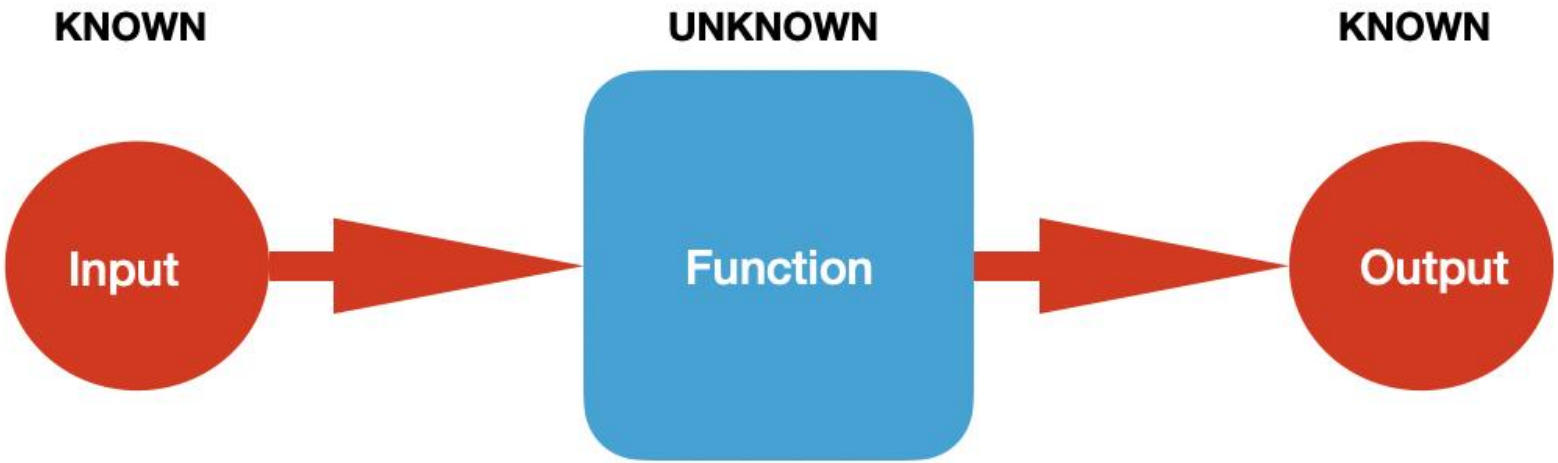
EURON

목차

Unsupervised Learning

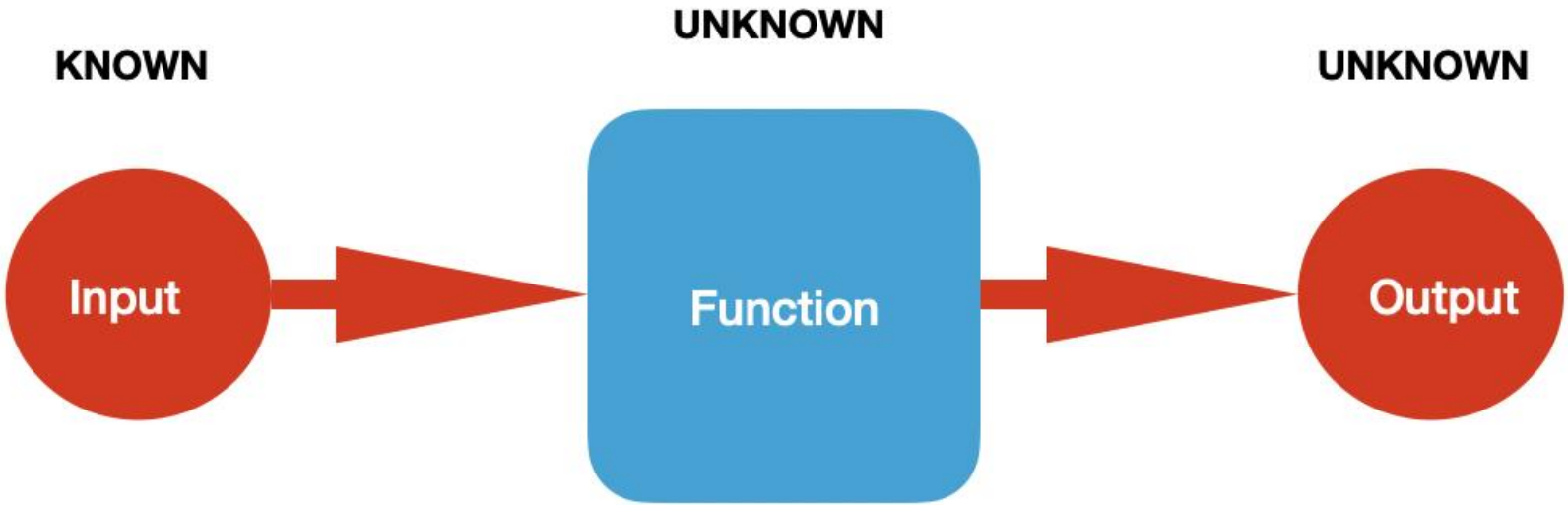
Supervised Learning

- Data X, Label Y
- X를 Y에 매핑하는 함수를 찾는 것
- Classification, object detection, semantic segmentation, image captioning



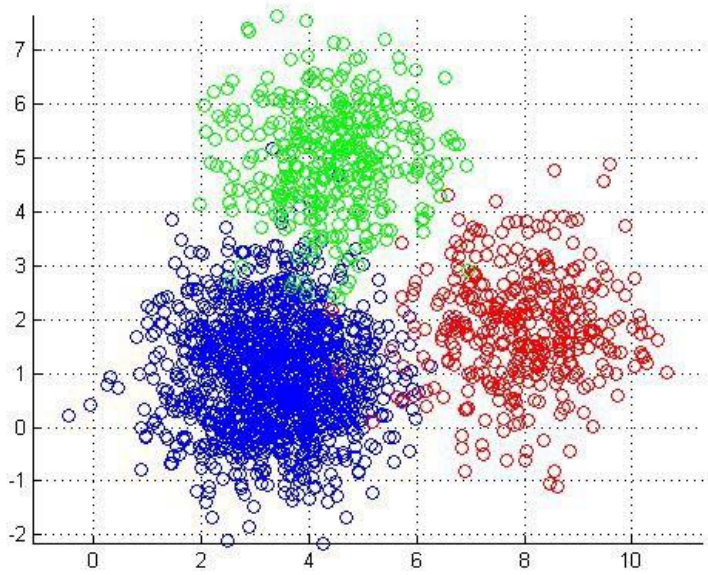
Unsupervised Learning

- Data의 기본적인 숨겨진 구조를 학습하는 것
- Label이 없는 input data
- Clustering, dimensionality reduction, feature learning, density estimation



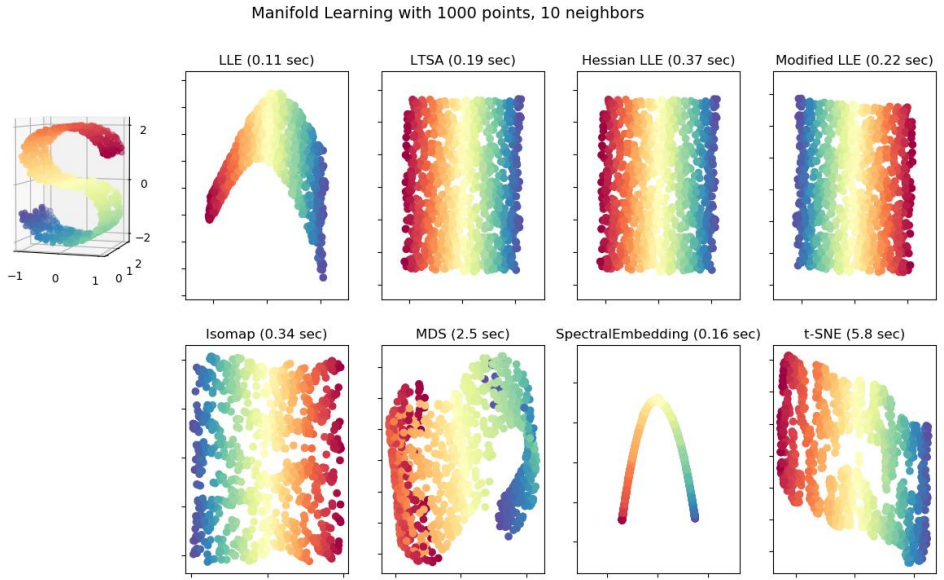
Unsupervised Learning

- Data의 기본적인 숨겨진 구조를 학습하는 것
- Label이 없는 input data
- Clustering, dimensionality reduction, feature learning, density estimation



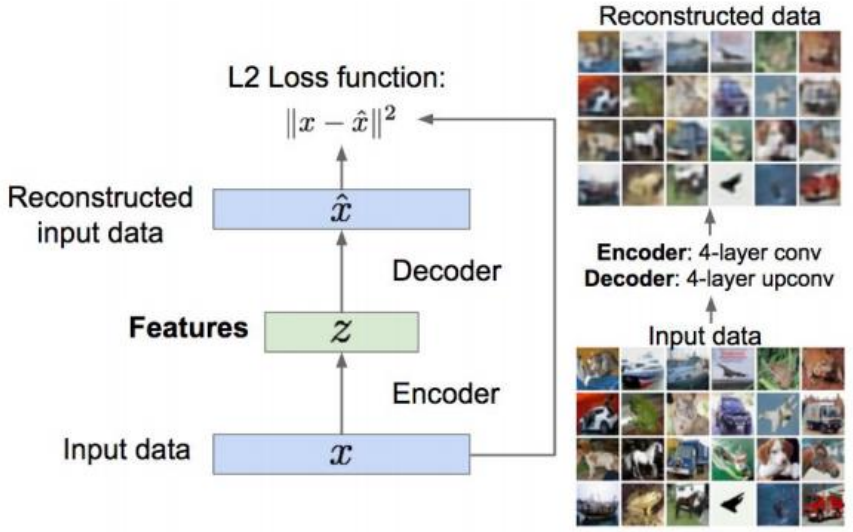
Unsupervised Learning

- Data의 기본적인 숨겨진 구조를 학습하는 것
- Label이 없는 input data
- Clustering, dimensionality reduction, feature learning, density estimation



Unsupervised Learning

- Data의 기본적인 숨겨진 구조를 학습하는 것
- Label이 없는 input data
- Clustering, dimensionality reduction, feature learning, density estimation

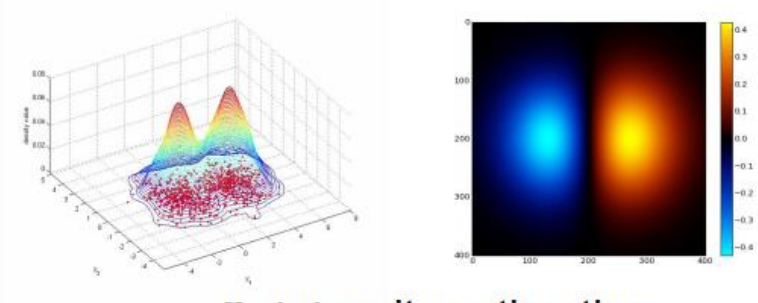


Unsupervised Learning

- Data의 기본적인 숨겨진 구조를 학습하는 것
- Label이 없는 input data
- Clustering, dimensionality reduction, feature learning, density estimation

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Unsupervised Learning

Generative Models

- 동일한 분포에서 새로운 sample을 생성해내는 것
- Super resolution, colorization, simulation with reinforcement learning, latent representation

Unsupervised Learning

Generative Models

- 동일한 분포에서 새로운 sample을 생성해내는 것

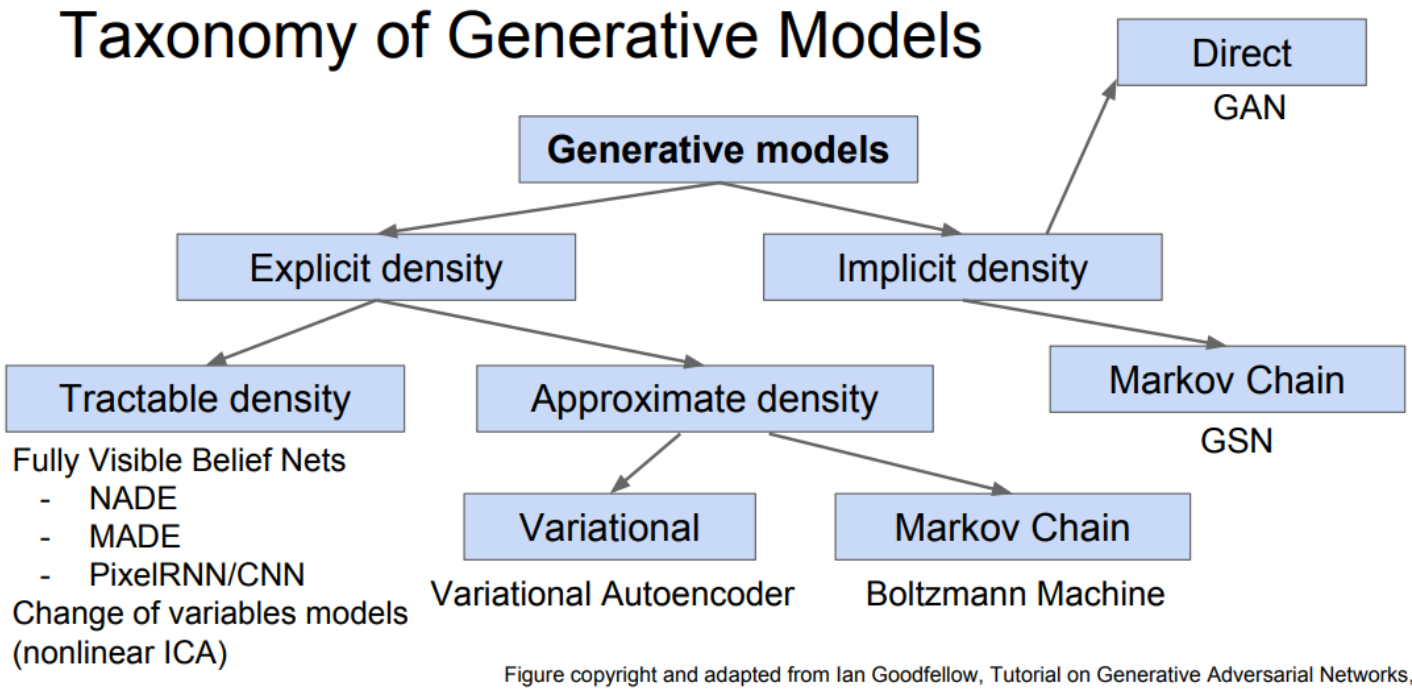


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN/CNN

Fully visible belief network

- Density를 명시적으로 정의하고 모델링함
- Chain Rule을 이용해 likelihood p(x)를 1차원 분포의 곱 형태로 분해
- 정의된 likelihood를 최대화 해야 함

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

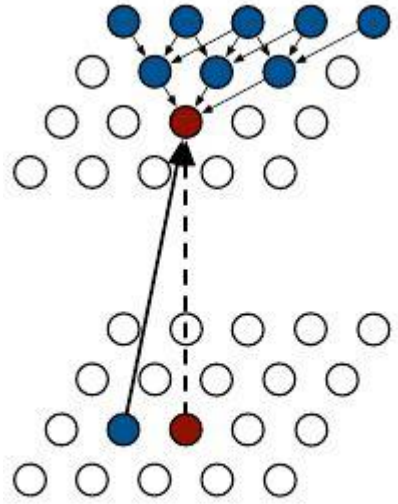
Likelihood of image x

Probability of i'th pixel value given all previous pixels

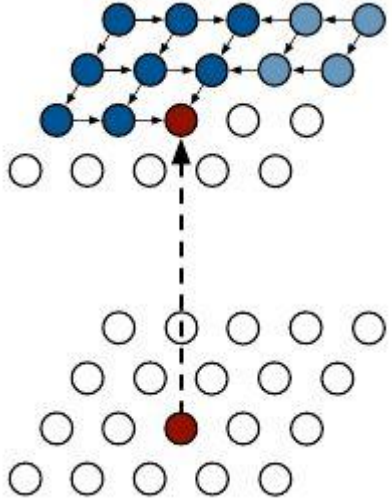
PixelRNN/CNN

PixelRNN

- Pixel 순서를 어떻게 다뤄야할 지의 문제를 해결하기 위해 고안된 방안
- 최대 12개의 fast 2D LSTM layer로 구성
  - Row LSTM
  - Diagonal BiLSTM



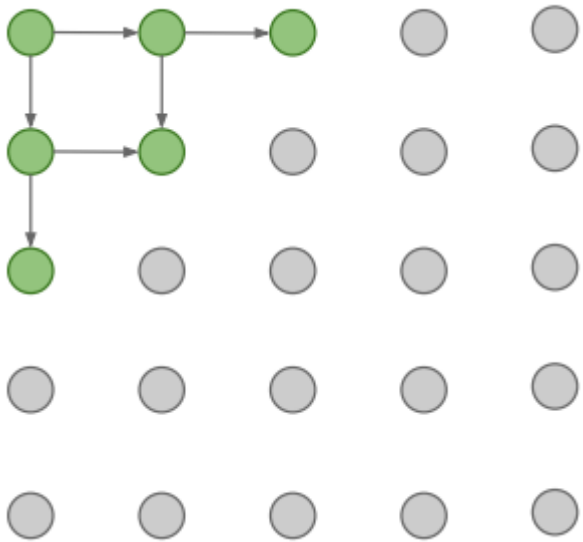
Row LSTM



PixelRNN/CNN

PixelRNN

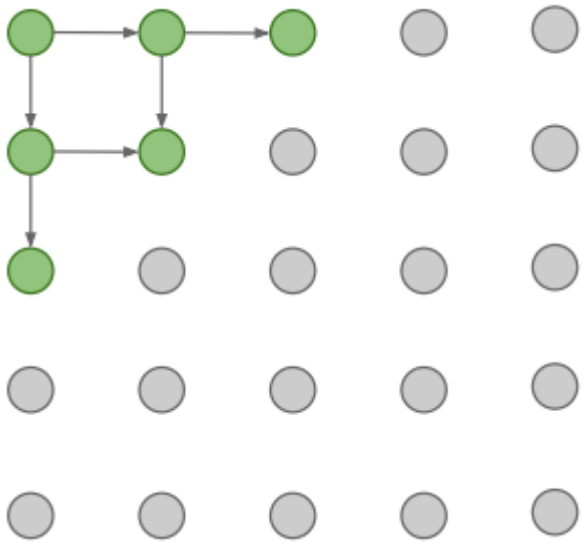
- 좌상단 코너에 있는 픽셀부터 시작
- 화살표 방향으로의 연결성을 기반으로 순차적으로 픽셀 생성 (RNN - 특히 LSTM - 이용)



PixelRNN/CNN

PixelRNN

- 좌상단 코너에 있는 픽셀부터 시작
- 화살표 방향으로의 연결성을 기반으로 순차적으로 픽셀 생성 (RNN - 특히 LSTM - 이용)

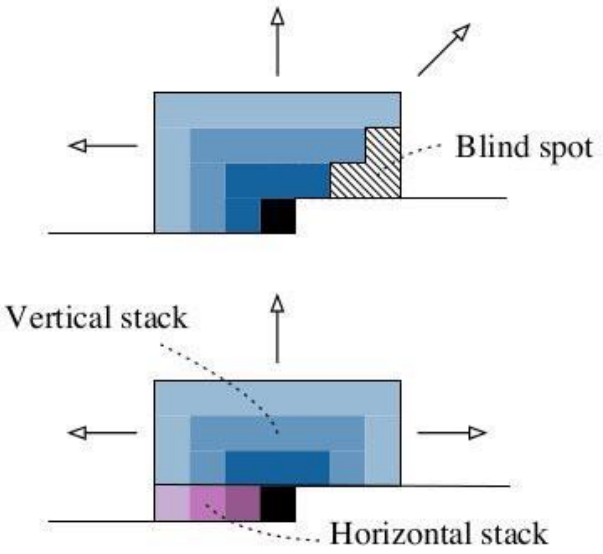


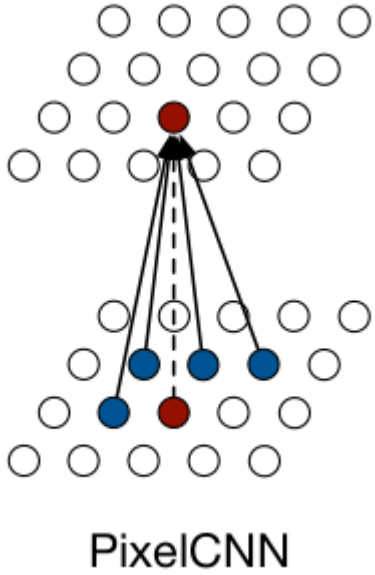
- 안정적으로 잘 동작함
- BUT, 순차적 생성방식
  - 새로운 이미지를 생성하고자 하면 여러 번의 feed forward 거쳐야 함
  - → 느림

PixelRNN/CNN

PixelCNN

- 기본 setting은 PixelRNN과 동일
- RNN 대신 CNN으로 modeling
- 픽셀을 생성할 때 특정 픽셀만을 고려함





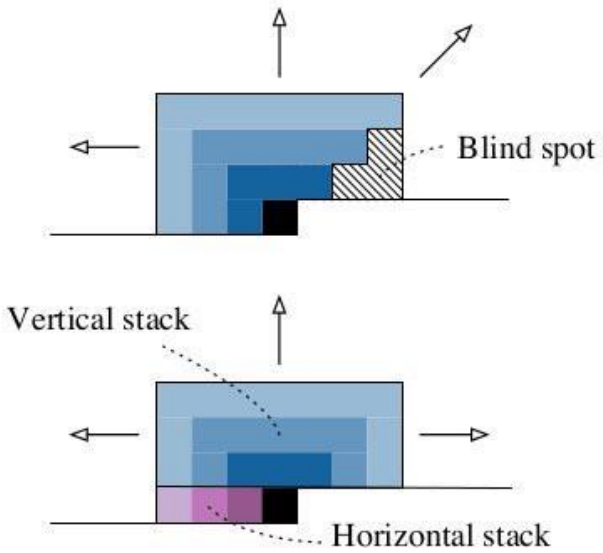
PixelCNN

PixelRNN/CNN

PixelCNN

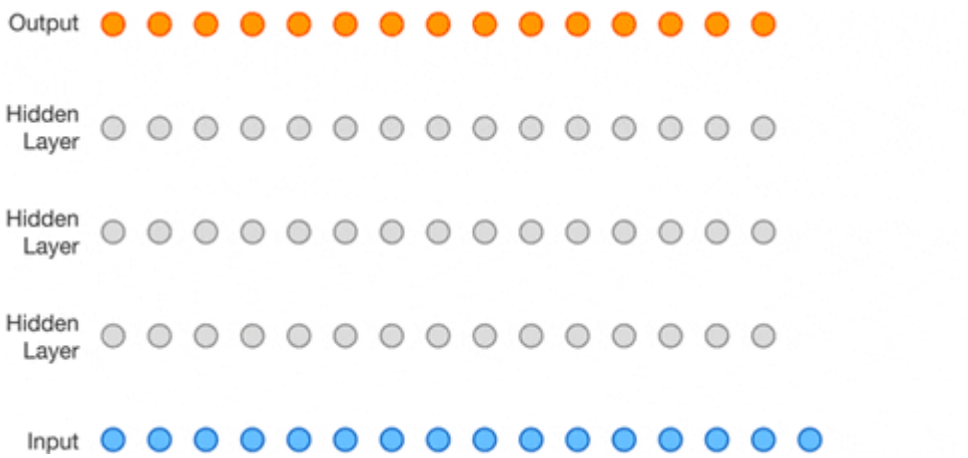
- 기본 setting은 PixelRNN과 동일
- RNN 대신 CNN으로 modeling
- 픽셀을 생성할 때 특정 픽셀만을 고려함

- PixelRNN에 비해 시간 단축
- 성능이 PixelRNN보다 나쁨
- Conv layer는 receptive field를 완전히 처리할 수 없으므로 픽셀 값이 조금씩 잘못 계산됨
- Blind spot 존재



PixelRNN/CNN

- Likelihood p(x)를 명시적으로 계산하는 방법
- Evaluation matric 존재
- BUT, 생성 과정이 순차적이기에 두 방식 모두 test phase에서는 느림
- 활용: 음성 생성



Variational Autoencoders (VAE)

PixelRNN/CNN

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

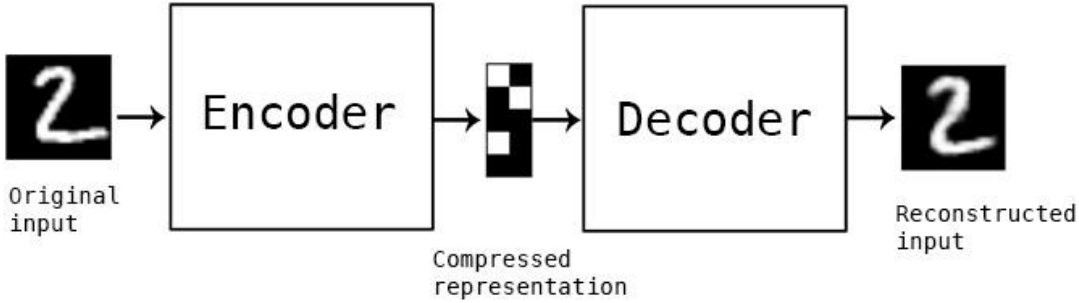
- Likelihood p(x)를 명시적으로 계산하는 방법
- 계산 가능한 확률 모델

VAE

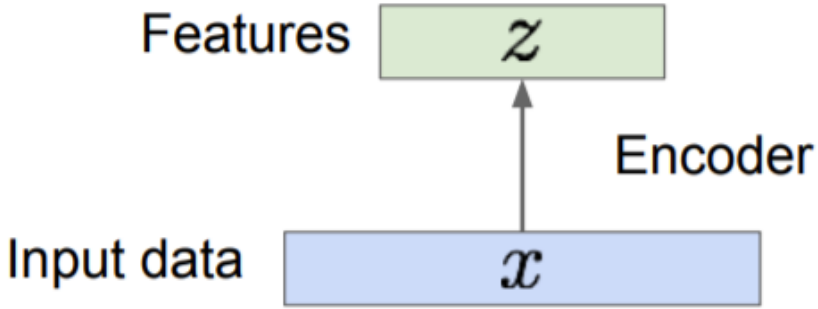
$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- 계산 불가능한 확률 모델 (intractable)
- 추가적 잠재 변수 z 모델링
- 적분 형태 →
  가능한 모든 z값에 대한 기댓값 구함
- 직접 최적화 불가

Autoencoders (AE)



Unsupervised Approach
For learning a lower-dimensional feature representation from unlabeled training data

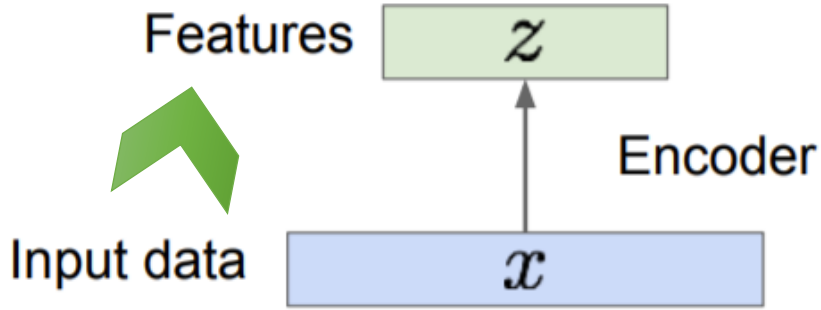


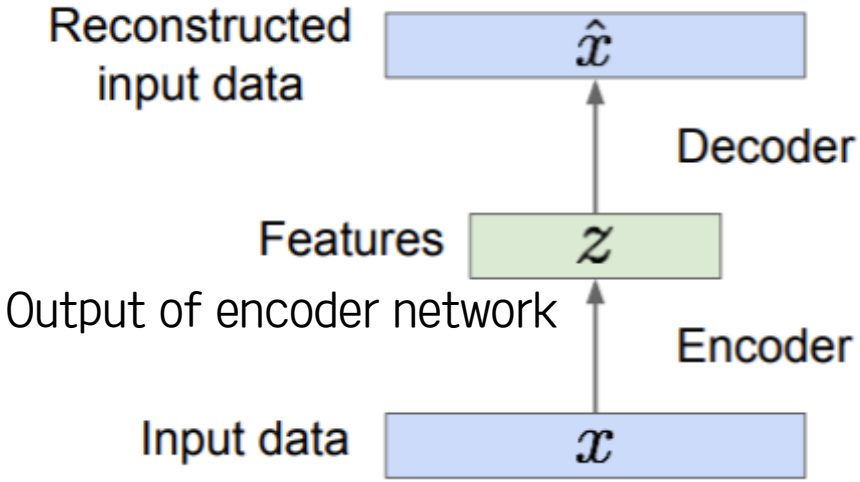
- Unlabeled training data → train low dimension feature representation
- Input data X → train some feature Z

Autoencoders (AE)

Dimensionality Reduction

차원 축소 – X의 중요한 feature들만 담기

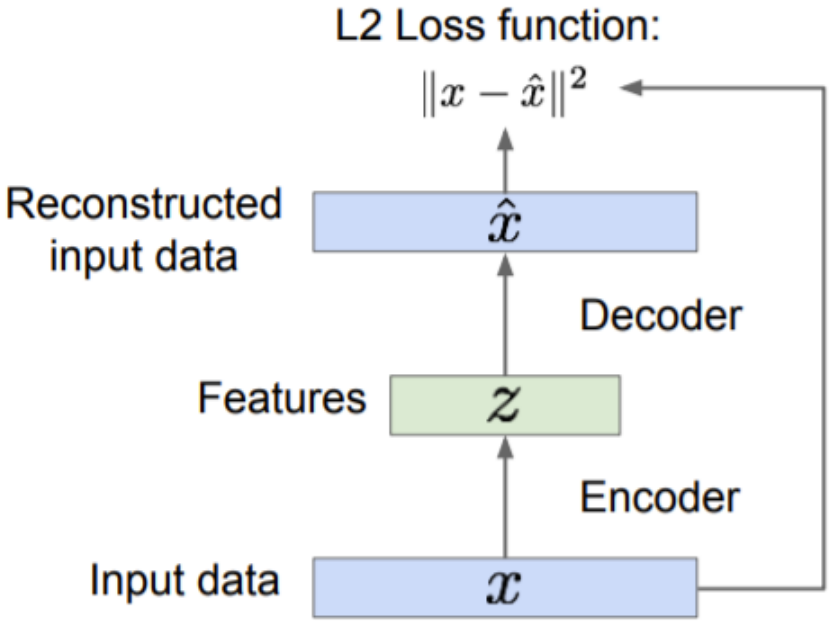


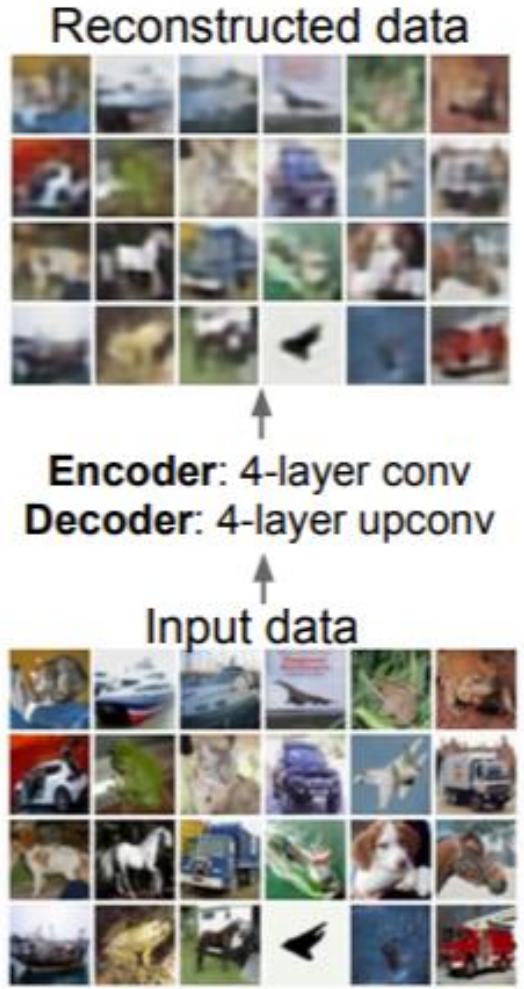


Autoencoding – Encoding itself

원본 데이터를 재구성하는 데에 필요한 feature들을 담기

Autoencoders (AE)





- Encoder: Conv l x → z l high → low
- Decoder: Upconv z → x l low → high
  - training 후에는 throw away

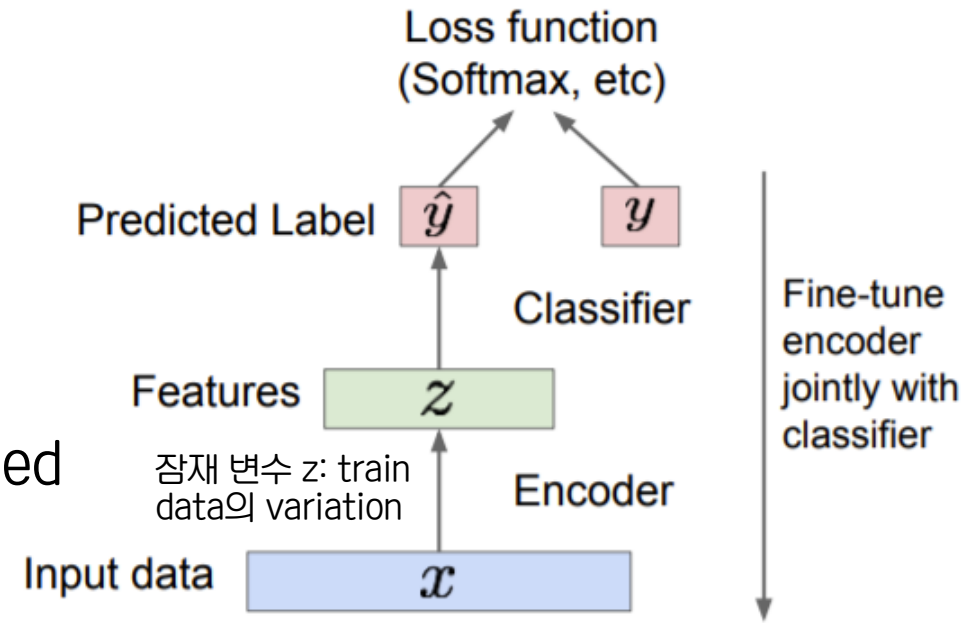
Autoencoders (AE)

AE의 목적

Unlabeling Data

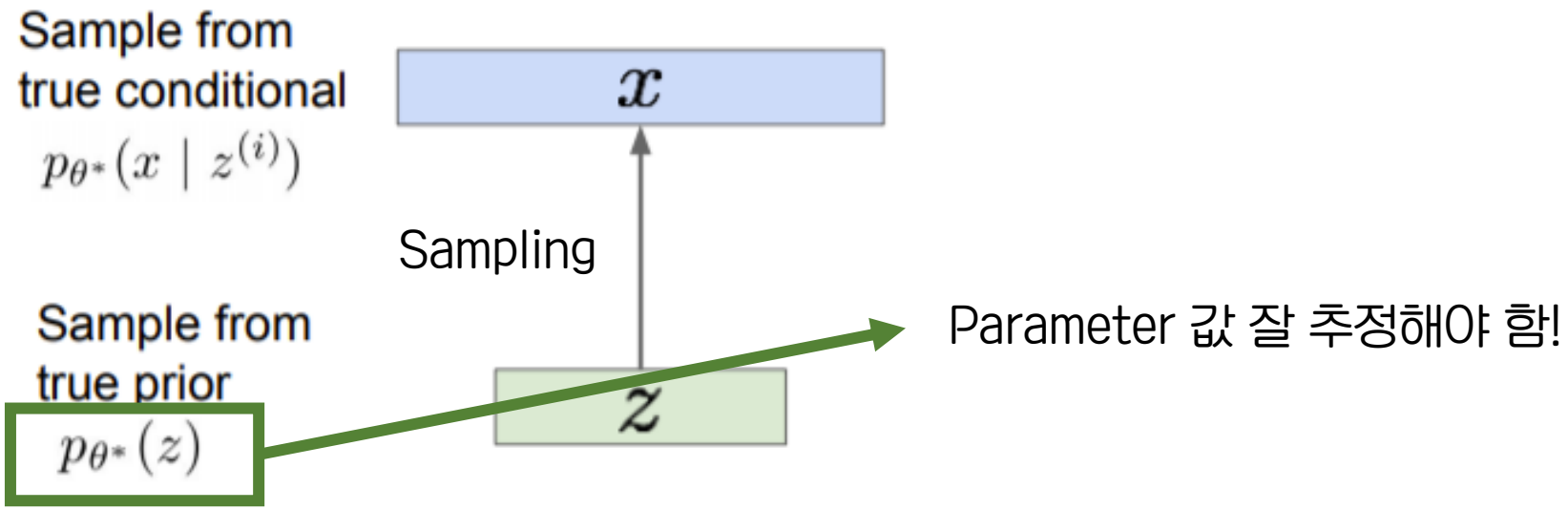
▶ GET 양질의 general feature representation

▶ USE 데이터가 부족한 supervised model의 initialize에 이용



"그렇다면 새로운 이미지를 생성해낼 수는 없을까?"

Variational Autoencoders (VAE)



Z: 다양한 속성을 담고 있는 잠재 요소

- 속성들이 어떤 distribution을 따르는지에 대한 prior(기댓값) 정의 필요

Variational Autoencoders (VAE)

VAE model의 training

P seta 정의

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

계산 불가!!!

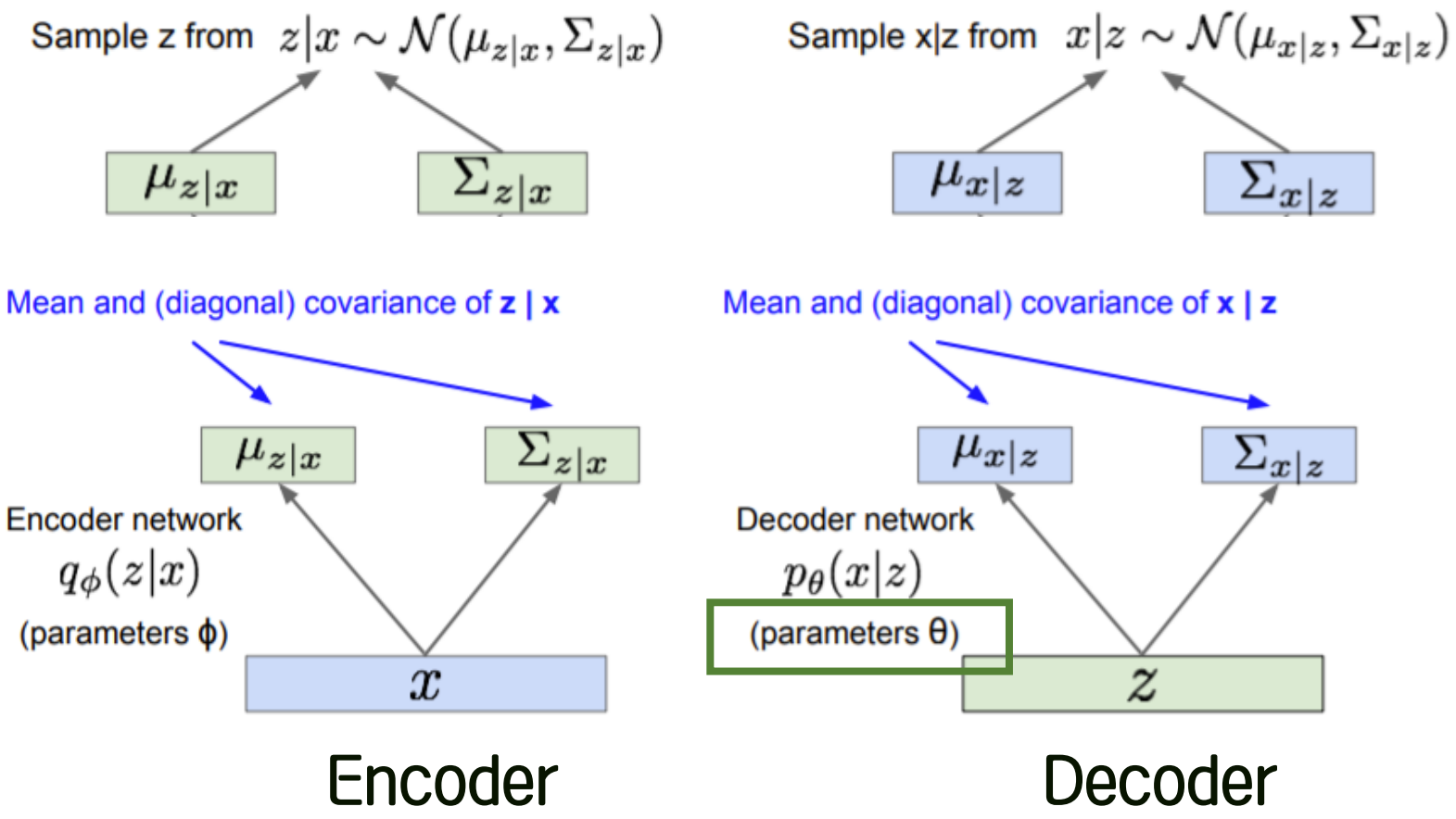
$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

$$p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x) \longrightarrow Q_\theta(x|z)$$

근사

→ lower bound 계산 가능

Variational Autoencoders (VAE)



Variational Autoencoders (VAE)

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)} \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)} \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)} \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

KL divergence

→ 두 분포 간의 거리를 측정하는 척도
항상 0보다 크거나 같으므로
계산이 불가하더라도 lower bound 알아낼 수 있음

Variational Autoencoders (VAE)

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

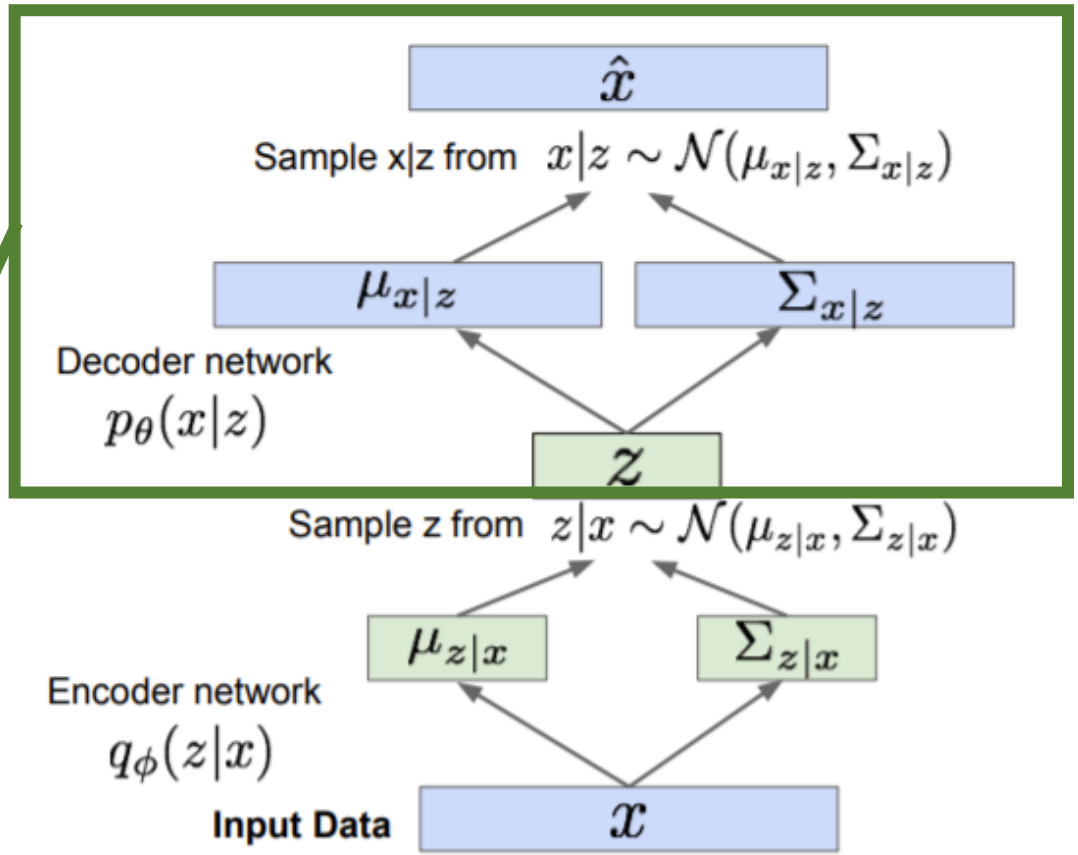
Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

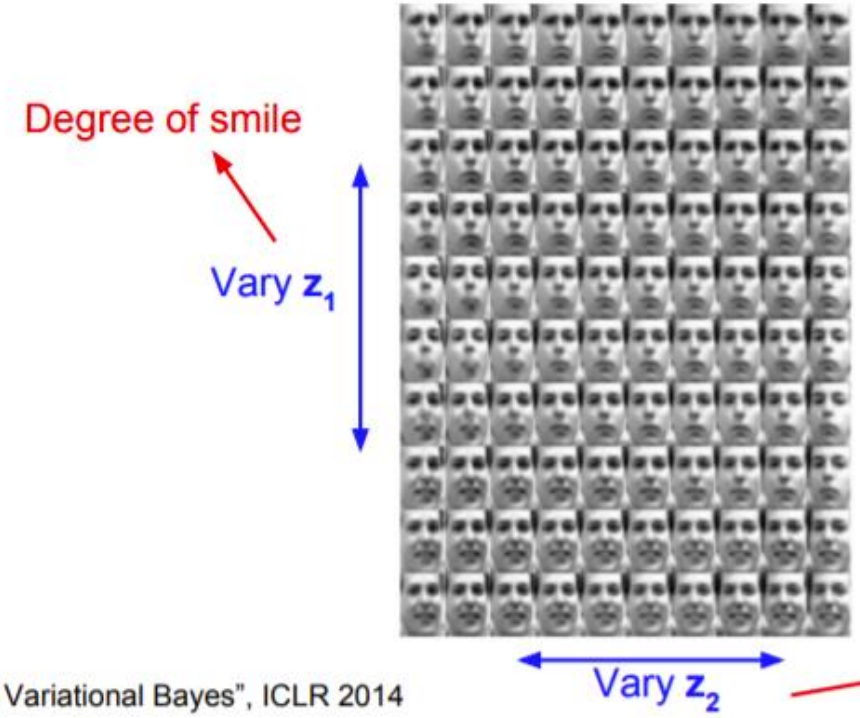
Variational Autoencoders (VAE)

VAE model의
training 과정



Test는 Decoder Network 이용

Variational Autoencoders (VAE)



g Variational Bayes", ICLR 2014

서로 독립되어 있는 잠재 변수 z
이용하여 다양한 이미지 제작 가능

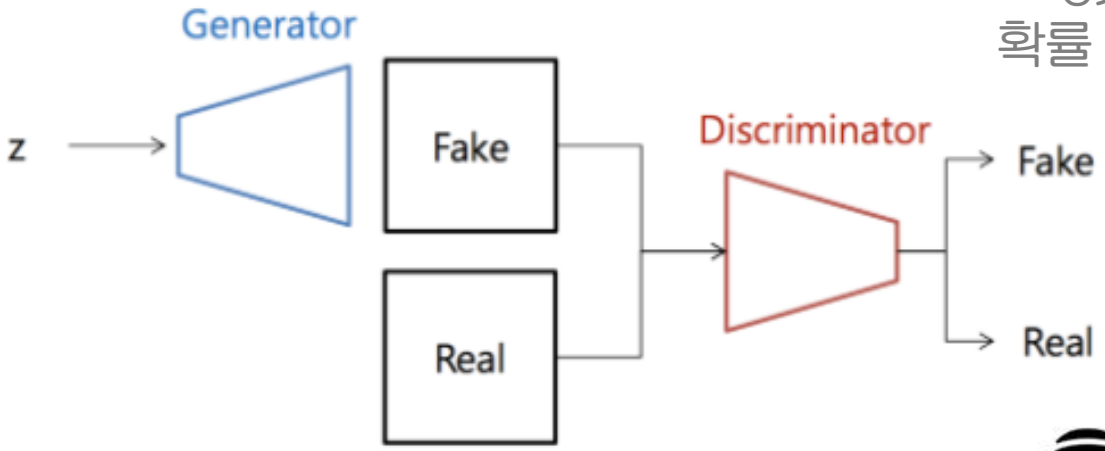
하지만 blur한 특징을 가짐

(이미지 퀄리티 ↓)

Labeled Faces in the Wild

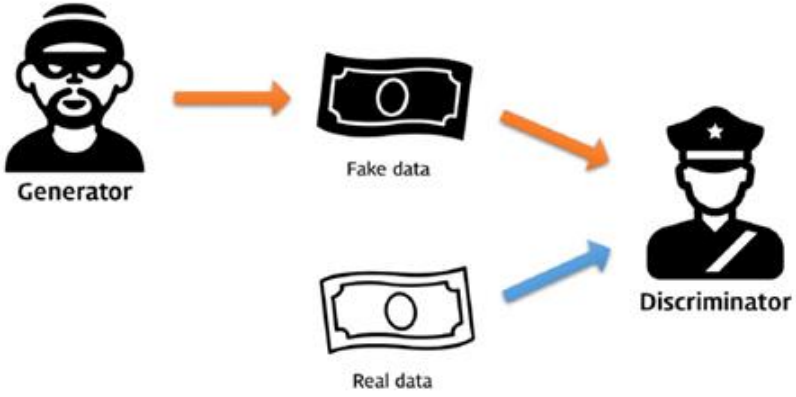
GAN

- GAN 구조와 원리



G가 D를 최대한 속일 수 있도록
확률 분포의 차이를 줄이는 것이 목적

*G : Generator
* D : Discriminator



Minimax two-player game ->

GAN

- PixcelCNN & VAE와의 차이점

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

VAEs define intractable density function with latent **z**:

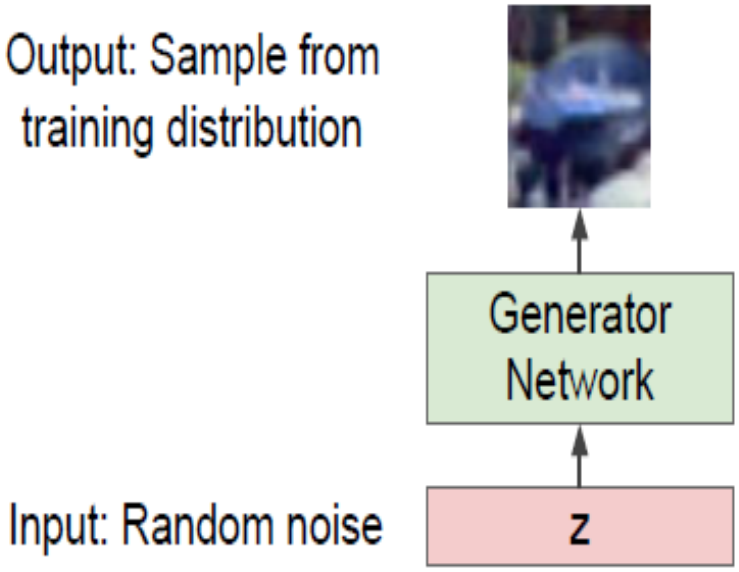
$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

PixcelCNN과 VAE와 달리 GAN에서는 P(X)를 수식으로 정의 X
>> 게임 이론 방식을 취하여 training distrivution을 학습함

GAN

- Input & Output



Input으로 gaussian random noise를 주고 neural network를 통해 training

GAN

- Adversarial nets

기호	설명
x	데이터
p_g	x에 대한 생성자의 분포
$p_z(z)$	input noise 변수
θ_g	multilayer perceptrions의 parameters
G	θ_g에 의해 표현되는 미분가능한 함수
$G(z;\theta_g)$	data space에 대한 mapping
$D(x)$	x가 p_g가 아니라 원본 데이터에서 나왔을 확률
$D(x;\theta_d)$	두 번째 multilayer perceptron

- Generator
  - generator's distribution p_g over data x 를 학습하기 위해서,
  - input noise variables p_z(z) 를 먼저 정의할 필요가 있고
  - data space로의 mapping을 G(z; θ_g)로 표현할 수 있다.
  - G는 미분 가능한 함수로, θ_g를 파라미터로 갖는 multilayer perceptron이다.
- Discriminator
  - D(x; θd) that outputs a single scalar (확률값이니까)
  - D(x) represents the probability that x came from the data rather than p_g.
- D to maximize the probability of assigning the correct label to both training examples and samples from G.
  We simultaneously train G to minimize log(1 – D(G(z)))

출처 : 연수님 GAN 논문 스터디 자료

GAN

- Minimax value function

* 식의 의미

- min_G: G는 V를 최소화하려고 한다.
- max_D: D는 V를 최대화하려고 한다. 2-player minimax 게임과 같으므로 당연하다.
- $\mathbb{E}$: 기댓값
- $x \sim p_{data}(x)$: x가 원본 데이터 분포에서 왔을 때

D(x) = 1일 때 최대

D(G(x)) = 0일 때 최대

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

>> **D 입장**(이상적인 Discriminator의 경우) 에서는 위 수식이 **0 -> Maximize**

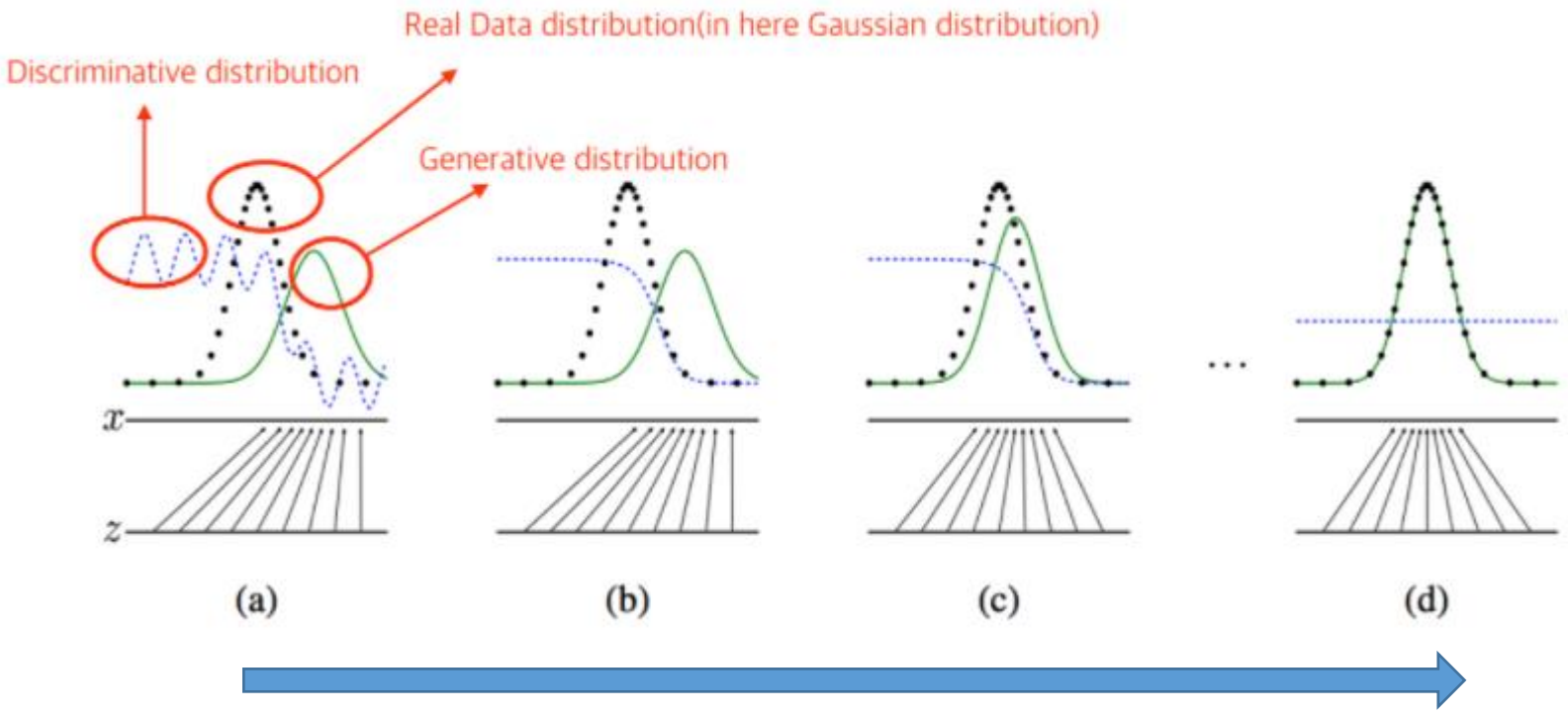
- D가 보는 sample x가 실제로 data (distribution)이라면 D(x)=1
  - 첫번째 term에서 log 값이 사라짐 => 0
- G가 만든 data라면 D(G(z))=0
  - 두 번째 term도 0이 됨

- 위 두 상황이 합쳐졌을 때 V의 최대값이 얻어지는 것

즉, D의 입장에서는 위의 수식이 0이 되어야 Maximize하는 것으로 볼 수 있다.

GAN

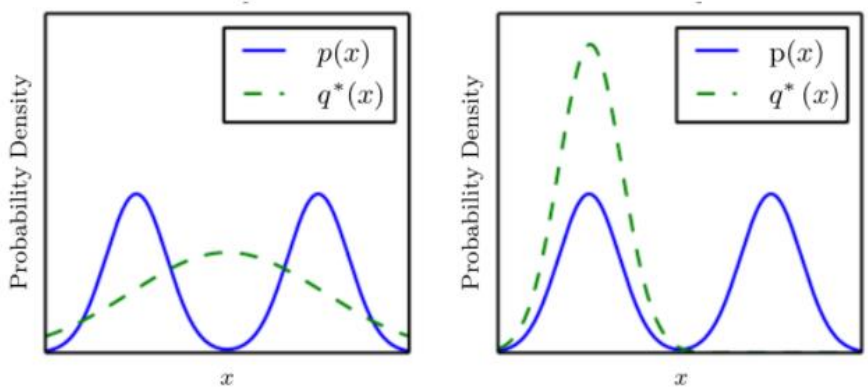
- Training



GAN

- 단점 및 해결방안

1. Mode collapsing



- Feature matching
- Mini-batch discrimination
- Historical averaging

GAN

- 단점 및 해결방안

2. Estimate

- 정성평가 : 사람이 직접 평가
- 학습된 classifier 이용
- Inception score : G가 생성한 데이터의 다양성(개성)을 측정하는 지표(클수록 good)

DCGAN :

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

> *"Most GANs today are at least loosely based on the DCGAN architecture."*
> *- NIPS 2016 Tutorial by Ian Goodfellow*

GAN의 불안정성(unstability)

- Approach of GAN : Minimax game
  - Theoretical ↔ In Practice

⇒ Deep Convolutional + GAN == DCGAN

DCGAN – Main Contribution

1. DCGAN은 **stable training**이 가능하다
2. 학습된 Discriminator는 **image classification 태스크에** 사용이 가능하다. (다른 unsupervised 알고리즘들과 비교할 것임.)
3. GAN이 학습한 **filters를 시각화할** 수 있고, 특정 오브젝트에 대해 **특정 filter를 학습했다는 점**을 보여줄 수 있다.
4. Generator는 **vector arithmetic properties**를 가지고 있다.

DCGAN - Architecture

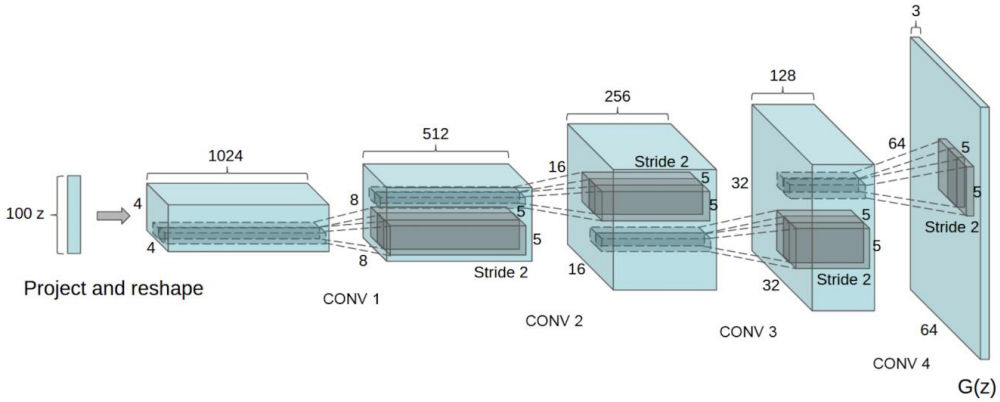


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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

DCGAN - Architecture

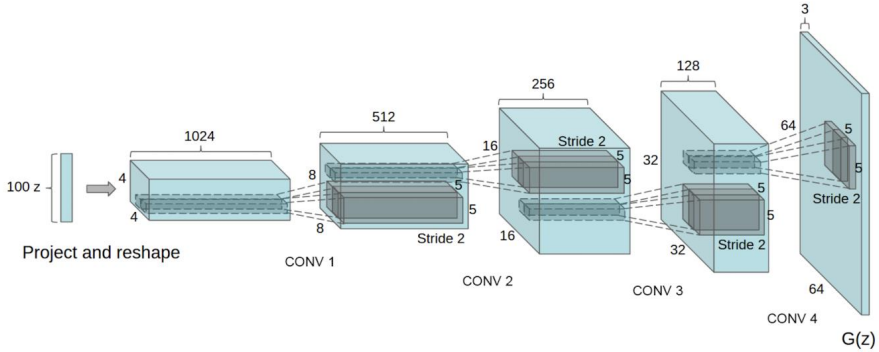


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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

DCGAN 이전에도, CNN을 이용한 GAN을 만드는 시도는 계속 있었음. ⇒ 만족스럽지 못했음.

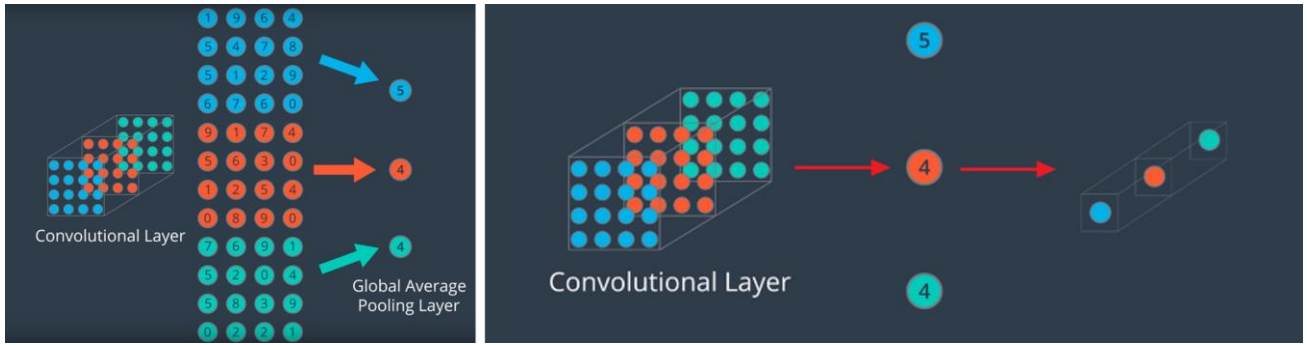
- DCGAN은 CNN architecture에서의
  최신 trend 3가지를 적용하여 성공시킴.
  ⇒ (Main Idea or Main Approach)

DCGAN - Main Idea (1)

- All Convolutional Net 사용
  - Striving for Simplicity: The All Convolutional Net 을 참고하여 All Convolutional Net을 사용
  - All conv. net
    - **pooling functions(e.g, max pooling)을 strided conv. 으로 바꾼 네트워크 구조**
  - 이 네트워크 구조를 통해 Generator와 Discriminator 모두 자신들의 spatial downsampling을 학습하기에 적합해짐.

DCGAN - Main Idea (2)

- Eliminating fully connected layers(FC layers) on top of convolutional features
  - 이 시점 trend는 마지막에 FC layer를 제거하고, global average pooling를 쓰는 방식이었다고 함.
  - 다만, 이 global average pooling은 **모델의 안정성은 올리는 반면에 convergence speed는 떨어뜨리는** trade-off 관계를 가지고 있음.



DCGAN - Main Idea (3)

- Batch Norm & ReLU activation
  - Batch Normalization는 학습을 안정화시킬 수 있는 방법
    - Review) Batch Norm → normalizing the input to each unit to have zero mean and unit variance
  - GAN Training의 문제점 해결에 도움을 줄 수 있음.
    - poor initialization로 인한 Training Problems
    - Deeper model에서 겪을 수 있는 gradient flow problems
  - **But, 모든 layer에 BN을 적용시키는 것은 아님.**
    - Generator의 output layer와 Discriminator의 input layer에는 BN을 적용시키지 **않음.**
  - Use ReLU activation function
    - Generator와 Discriminator에 적용되는 function이 다름.
    - Generator → 기본적으로 ReLU를 사용, output layer에는 Tanh function 사용
    - Discriminator → Leaky ReLU 사용

DCGAN - Main Idea + Code Implementation

```python
class Generator(nn.Module):
    def __init__(self, latent_dim = 100, out_channel=3):
        super(Generator, self).__init__()

        self.conv_block = nn.Sequential(
            nn.ConvTranspose2d(in_channels=latent_dim, out_channels=512, kernel_size=4, stride=1, padding=0),
            nn.BatchNorm2d(num_features=512),
            nn.ReLU(True),

            nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(num_features=256),
            nn.ReLU(True),

            nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(num_features=128),
            nn.ReLU(True),

            nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.ConvTranspose2d(in_channels=64, out_channels=out_channel, kernel_size=4, stride=2, padding=1),
            nn.Tanh()
        )

    def forward(self, x):
        x = self.conv_block(x)
        return x
```

```python
class Discriminator(nn.Module):
    def __init__(self,in_channel=3):
        super(Discriminator, self).__init__()

        self.conv_block = nn.Sequential(
            nn.Conv2d(in_channels=in_channel, out_channels=64, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(in_channels=512, out_channels=1, kernel_size=4, stride=1, padding=0),
            nn.Sigmoid()
        )


    def forward(self, x):
        x = self.conv_block(x)
        return x
```

DCGAN – Main Idea + Code Implementation

https://github.com/yskim0/Deep-Convolutional-GAN

DCGAN - Details of Adversarial Training

- Generation vs. Memorization
  - ○ Memorization : generated image가 학습 데이터에서 기억하여 만들어진 것이라면 진정한 Generation이 아님.
- ● To avoid memorization → “Image de-duplication process”
  - ○ Use an Autoencoder
  - ○ *LSUN의 경우 Deduplication이라고 해서 혹시나 생길지 모르는 1번 문제 즉, memorization 문제를 피하기 위해서 이미지들을 Autoencoder를 이용하여 짧은 code로 변환할 수 있도록 학습한 후 이 code들에서 가까운 값들을 제거 했다고 합니다. 비슷한 이미지들을 약 275,000개 제외할 수 있었다고 하네요. 기술적인 내용이긴 하지만 논문을 보다가 무슨 소린가 할 수 있는 부분이라 부연해보았습니다. (출처: http://jaejunyoo.blogspot.com/2017/02/deep-convolutional-gan-dcgan-2.html)*

DCGAN - Empirical Validation

1. DCGAN은 stable training이 가능하다
2. 학습된 Discriminator는 image classification 태스크에 사용이 가능하다. (다른 unsupervised 알고리즘들과 비교할 것임.)
3. GAN이 학습한 filters를 시각화할 수 있고, 특정 오브젝트에 대해 특정 filter를 학습했다는 점을 보여줄 수 있다.
4. Generator는 vector arithmetic properties를 가지고 있다.

Feature Extractor로써의 역할이 가능한가 → classification task도 잘 수행할 수 있는가

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

Table 2: SVHN classification with 1000 labels

Model	error rate
KNN	77.93%
TSVM	66.55%
M1+KNN	65.63%
M1+TSVM	54.33%
M1+M2	36.02%
SWWAE without dropout	27.83%
SWWAE with dropout	23.56%
DCGAN (ours) + L2-SVM	22.48%
Supervised CNN with the same architecture	28.87% (validation)

DCGAN - Visualizing the internals of the networks

1. DCGAN은 stable training이 가능하다
2. 학습된 Discriminator는 image classification 태스크에 사용이 가능하다. (다른 unsupervised 알고리즘들과 비교할 것임.)
3. GAN이 학습한 filters를 시각화할 수 있고, 특정 오브젝트에 대해 특정 filter를 학습했다는 점을 보여줄 수 있다.
4. Generator는 vector arithmetic properties를 가지고 있다.

Random filters **Trained filters**

discriminator가 feature들을 학습해서, 특정 파트(bed, windows,…)들에 대하여 active 하고 있음을 볼 수 있음.

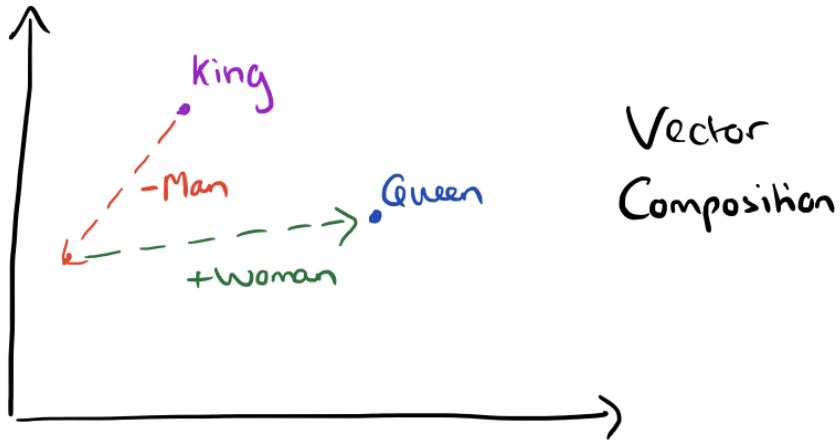
Figure 6: Top row: un-modified samples from model. Bottom row: the same samples generated with dropping out "window" filters. Some windows are removed, others are transformed into objects with similar visual appearance such as doors and mirrors. Although visual quality decreased, overall scene composition stayed similar, suggesting the generator has done a good job disentangling scene representation from object representation. Extended experiments could be done to remove other objects from the image and modify the objects the generator draws.

Generator가 무슨 representation을 학습했는지 알아보기 위하여 특정 filter(여기서는 window filter)를 삭제해봄.

- 이는 즉, Window라는 object를 **Forget** 하는 것. (기술적으로는 window filter를 dropout 시킨다.)

결과적으로 이 실험에서는 창문이 아닌 다른 representations, objects가 들어감.

DCGAN - Vector Arithmetic

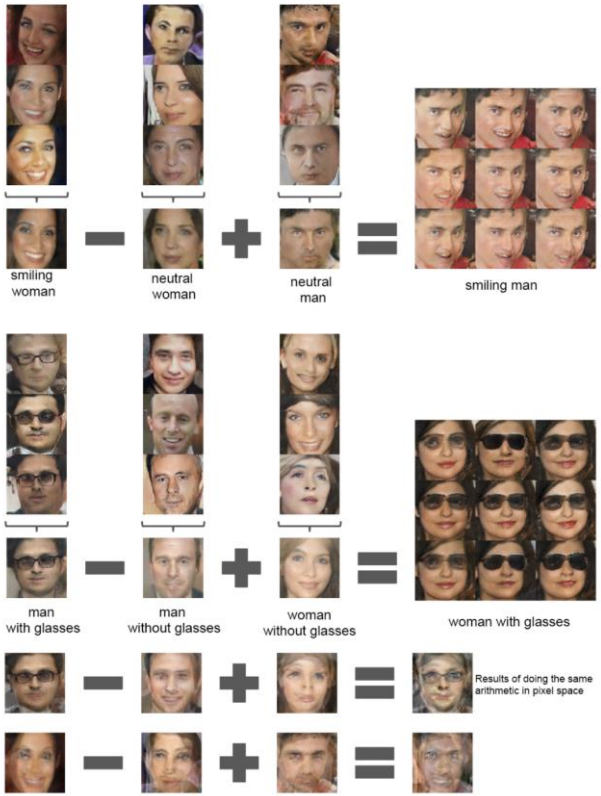


https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

word embedding 으로 vector("King") - vector("Man") + vector("Woman")
가 vector("Queen")의 결과가 나오듯이, DCGAN에서도 이와 비슷한
arithmetic한 연산이 가능하다.

DCGAN - Vector Arithmetic

Generator의 input인 Z vector에 대한 arithmetic operation을 하는데,
single sample로는 불안정하여 **3개의 Z vector를 평균한 값을 사용**

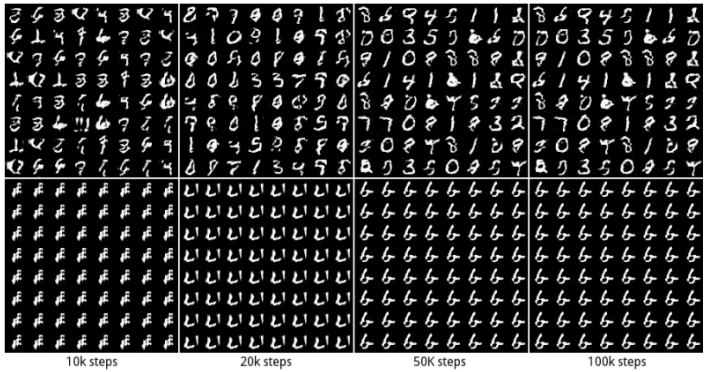


smiling woman - neutral woman + neutral man = smiling man

DCGAN - Future Work

stability를 완전히 해결한 것이 아님.

- DCGAN을 오랫동안 학습하게 되면 mode collapse, oscillating mode가 발생할 수 있음
  → 아직도 **불안정한 학습**



Recap

Recap

Generative Models

- PixelRNN and PixelCNN — Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.
- Variational Autoencoders (VAE) — Optimize variational lower bound on likelihood. Useful latent representation, inference queries. But current sample quality not the best.
- Generative Adversarial Networks (GANs) — Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.

Also recent work in combinations of these types of models! E.g. Adversarial Autoencoders (Makhanzi 2015) and PixelVAE (Gulrajani 2016)

감사합니다

Q&A