exam 개인학습

Lec 12

AI summary Try on this page

Type something...



# 01) Generative Models

### 01_Supervised vs Unsupervised Learning

## Supervised vs Unsupervised Learning

**Supervised Learning**

**Data**: (x, y)
x is data, y is label

**Goal**: Learn a *function* to map x -> y

**Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

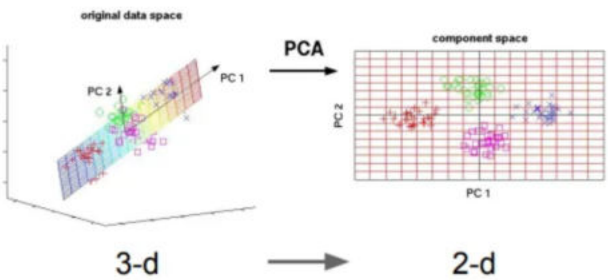
- Supervised(지도) 학습은 학습 데이터와 label를 주고 데이터와 label를 mapping하는 함수를 찾는 것이다.
- 지금까지 배운 대부분의 것이 Supervised이다.

## Supervised vs Unsupervised Learning

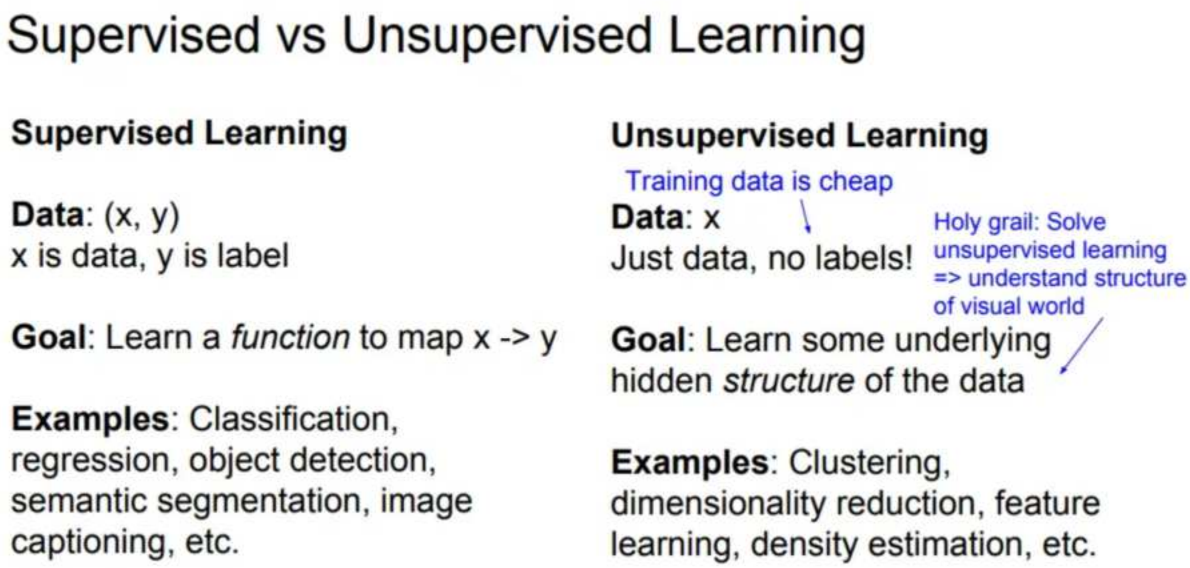
**Unsupervised Learning**

**Data**: x
Just data, no labels!

**Goal**: Learn some underlying hidden *structure* of the data

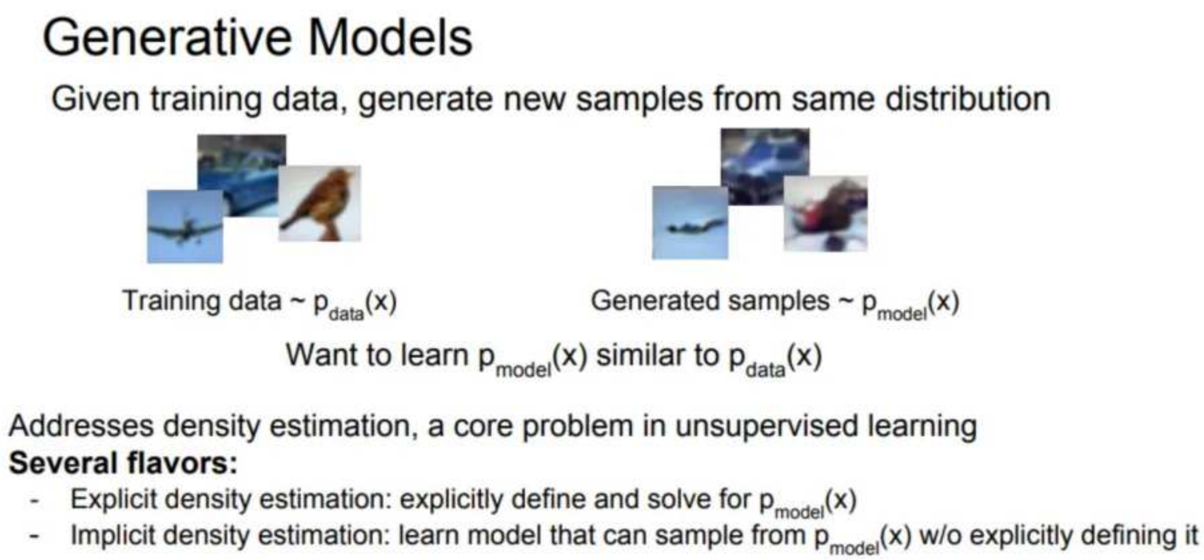



- Unsupervised(비지도) 학습은 label은 없고 그냥 데이터만 주어진다.
- 목표는 데이터의 숨겨진 구조를 찾는 것이다.
- 대표적인 예시로는 Clustering, dimensionality reduction, feature learning, density estimation 등이 있다.

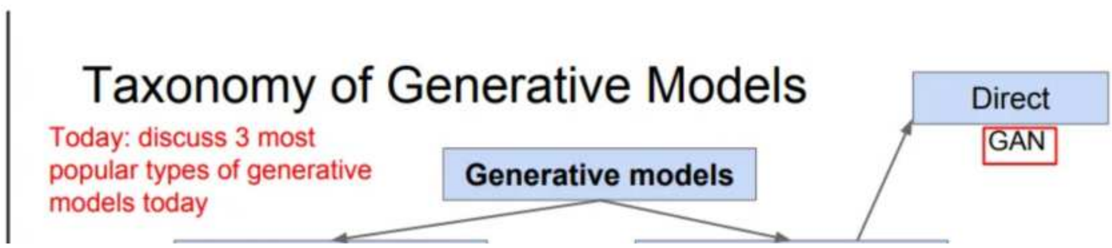


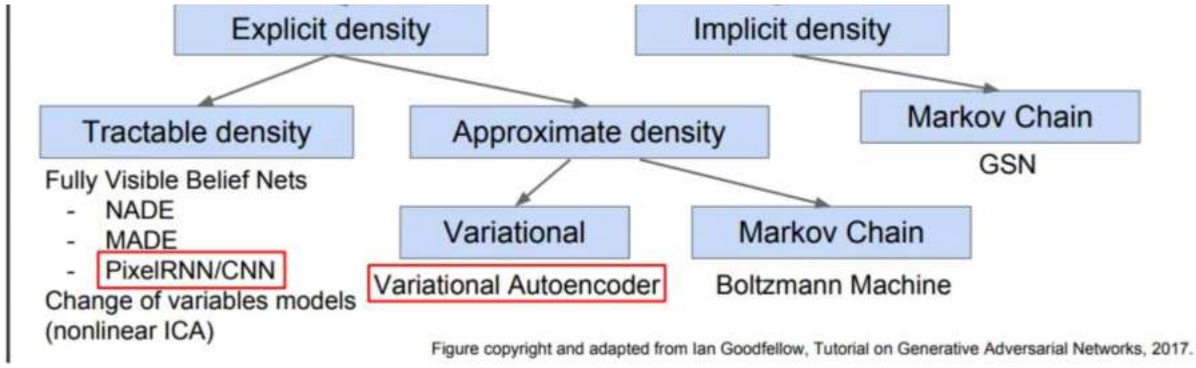

- 비지도학습의 경우 레이블이 없기 때문에 데이터가 매우 싸서 아주 많이 모을 수 있다.
- 하지만 아직까지 open problem이 많이 존재한다.

### 02_Generative Models




- train data가 주어졌을 때 그것과 비슷한 분포를 지니는 new data를 생성하는게 생성모델의 역할이다.

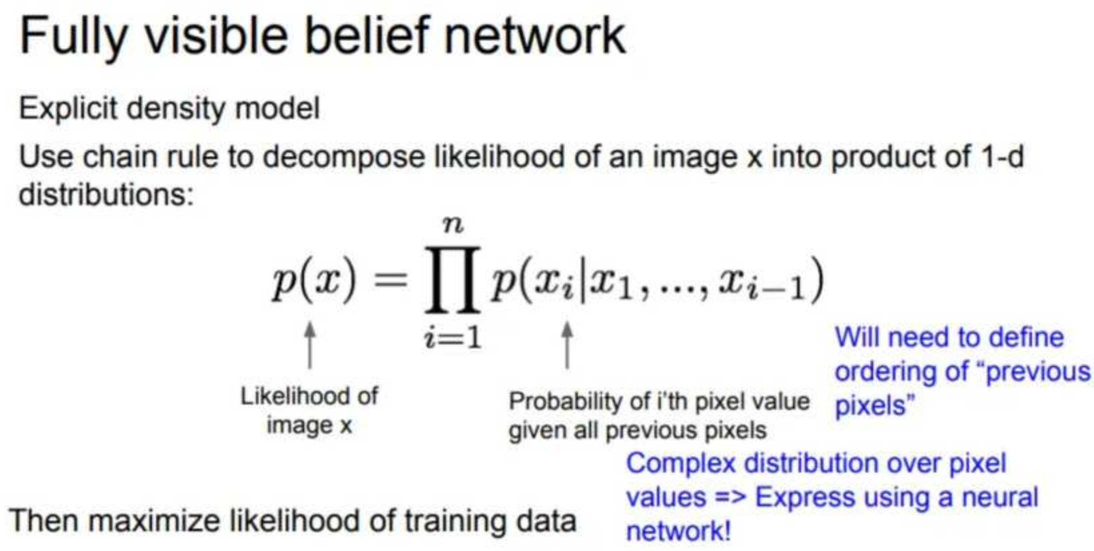





taxonomy: 분류학

- Explicit density는 가우시안 같이 확률 모델인 density function을 정의하는 것이다.
- Explicit density는 모델링 하는 density function이 명확하다.
- Tractable density는 gradient를 구해서 직접적으로 학습한다.
- Variational Autoencoder(VAE)는 objectives의 gradient를 구해서 최적화하는 것이 아니라, 근사치를 사용하기 때문에 Approximate density라고 부른다.
- GAN은 sampling을 사용하지 않기 때문에 Implicit density라고 불린다.

### 3. Tractable density model(PixelRNN/CNN)




- p(x) 이 어떤 분포를 띄는지를 정의하고 찾는데 초점을 둔다.
- Explicit density 모델은 training data의 likelihood를 높이는 방향으로 학습을 한다.
- 모델을 학습시키려면 likelihood를 최대화시키면 되는데, 픽셀 값의 분포가 매우 복잡하다.
- 따라서 이미지 내의 각 픽셀들의 분포를 알고 복잡한 함수를 표현하기 위해서 신경망을 이용할 것이다.

#### 3.1 Likelihood & Maximum Likelihood Estimation(MLE)

$$X \sim P_\theta(X) \cdots \text{확률변수}X\text{가 모수 }\theta\text{에 대해 가지는 분포}$$

$$\mathcal{L}(\theta|x) = Pr(X = x|\theta)$$
$$\mathcal{L}(\theta|x) = \text{가능도 함수}$$

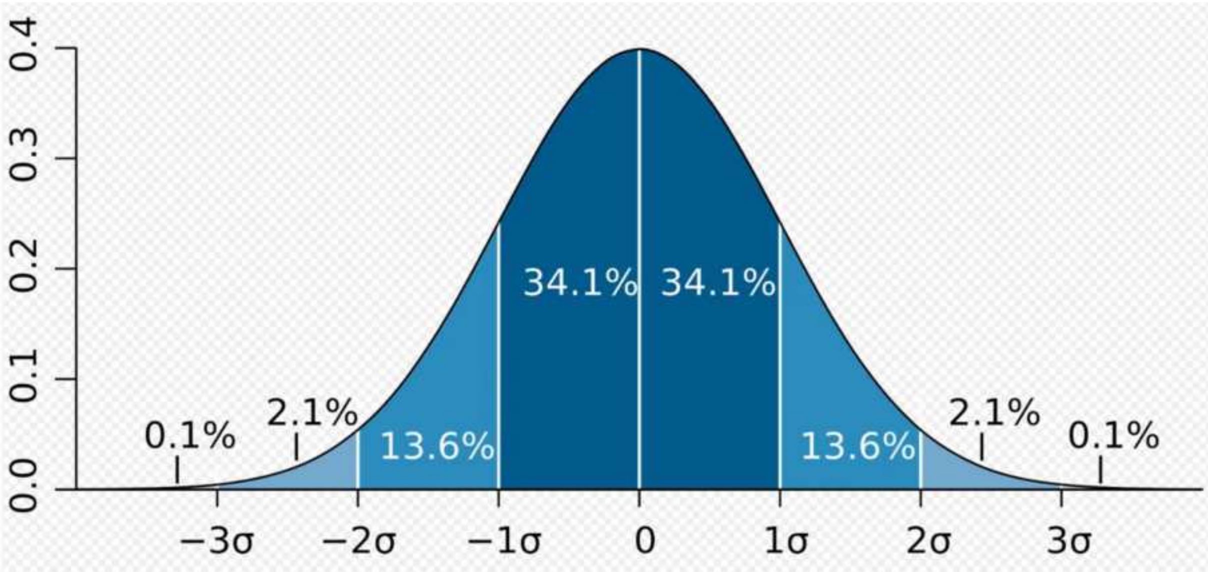
$$Pr(X = x|\theta) = Pr(x_1, x_2, x_3 \cdots|\theta) = Pr(x_1|\theta) \times Pr(x_2|\theta) \times \cdots Pr(x_n|\theta)$$
$$\mathcal{L}(\theta|x) = Pr(x_1|\theta) \times Pr(x_2|\theta) \times \cdots Pr(x_n|\theta)$$

- chain rule에 의해 **조건부확률들의 곱으로 분해할 수 있는데** 각각 **p(x_i | conditions)를 정의할 수 있다.**
- 결국 알고리즘은 **cross entropy를 최소화 하는 방향으로 동시에 likelihood를 최대화 하는 방향**으로 학습을 진행한다.

- 확률: pdf의 면적
- 가능도(likelihood, 가능도): PDF의 y값 = 확률밀도

$$Likelihood = L(\theta|D)$$

- θ : parameter, D: data
- Likelihood란 관측값 data가 주어졌을 때(given) 관측값이 θ에 대한 **확률분포 P(θ)**에서 나왔을 **확률**이다.

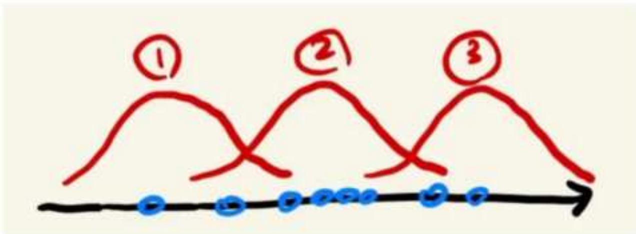



- 확률은 주어진 확률분포에서 해당 관측값이 어느정도 나올지를 표현하고, 이 값은 distribution의 area를 통해 구한다.
- 반면 가능도(Likelihood)는 관측 데이터가 있을 때 어떤 분포를 주고 그 분포에서 데이터가 나왔을 확률을 구하는 것이다.

$$L(\theta|x) = P_\theta(X = x) = \prod_{k=1}^{n} P(x_k|\theta)$$

- 즉, 가능도는 각 데이터 샘플에서 후보 분포 P(θ)에 대한 y value(밀도)를 (데이터의 추출이 독립적이고 연달아 일어나는 사건이므로) 곱하여 계산한다.
- 직관적으로 생각했을 때 후보가 되는 분포가 데이터를 잘 설명한다면 likelihood는 당연히 높게 나올 것이다.
- 보통은 likelihood function을 대체하여 log-likelihood function을 사용한다.

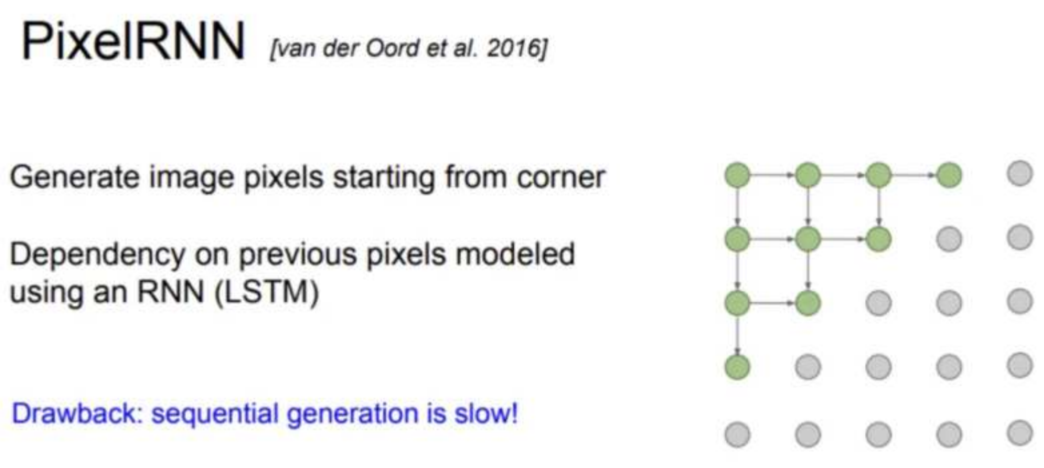
$$L(\theta|x) = log(P_\theta(X = x)) = log\prod_{k=1}^{n} P(x_k|\theta)$$




- 그림처럼 파란색 데이터의 분포를 설명하는 후보 분포가 1, 2, 3으로 주어졌다고 가정해보자.
- 각 분포에 대한 likelihood를 구하면 2번이 가장 큰 값을 가지게 될 것이다.
- 이렇게 가장 데이터를 잘 설명하는 분포를 likelihood를 통해 구할 수 있고 이런 방법을 Maximum Likelihood Estimation(MLE)라고 부른다.
- 최대값은 구하기 위해서는 θ에 대하여 편미분을 해서 0인 지점을 찾는 것이다.

$$\frac{\partial}{\partial\theta}L(\theta|x) = \frac{\partial}{\partial\theta}logP(x|\theta) = \sum_{i=1}^{n}\frac{\partial}{\partial\theta}logP(x_i|\theta) = 0$$

3.2 Pixel RNN



- PixelRNN은 왼쪽 위 코너를 시작점으로 상하좌우로 뻗어나가면서 이미지를 pixel by pixel로 생성하는 방법이다.
- 새로 만들어지는 pixel은 인접한 pixel의 영향을 받아서 새로 상성된다.
- 이전 결과에 영향을 받는 구조이기 때문에 LSTM 같은 RNN 구조를 사용한다.
- 단점으로는 순차적으로 생성하기 때문에 시간이 매우 오래 걸린다는 것이다.

### 3.3 Pixel CNN

## PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training is faster than PixelRNN (can parallelize convolutions since context region values known from training images)

Generation must still proceed sequentially => still slow

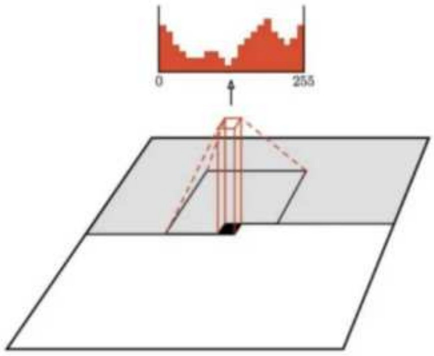



Figure copyright van der Oord et al., 2016. Reproduced with permission.

- RNN을 CNN으로 대체한 방법이다.
- 인접한 좌표들에 한꺼번에 CNN을 적용시켜 PixelRNN보다 빠르다
- PixelCNN & PixelRNN 모두 explicit 하게 분포를 정의한다.
- 하지만 여전히 느리다.

### 04_Variational Autoencoders(VAE)

## So far...

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**:

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

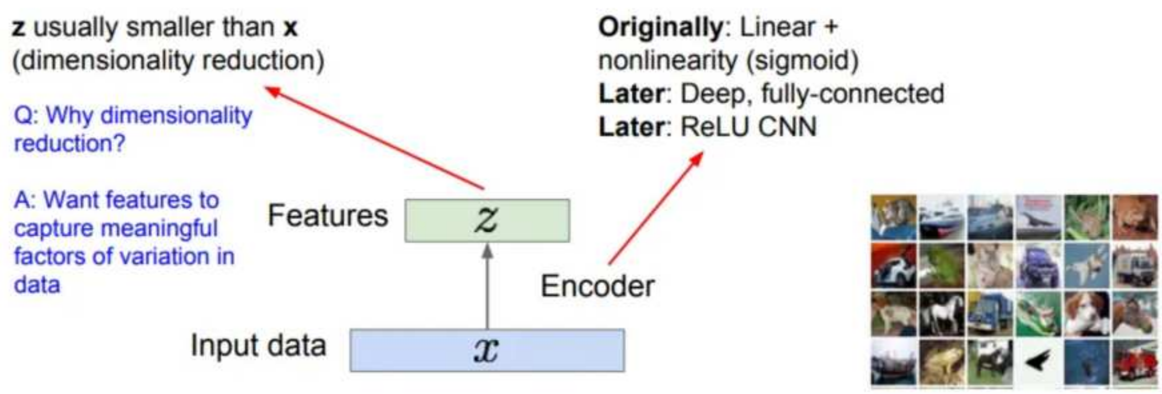
Cannot optimize directly, derive and optimize lower bound on likelihood instead

- VAE는 앞 모델들과 다르게 직접 계산이 불가능한(intractable) 확률 모델을 정의한다.
- 추가적인 잠재 변수(latent variable) z를 모델링할 것이다.
- p(x)가 적분의 형태를 띄고 있어서 직접 최적화시킬 수는 없다.
- 대신에 likelihood(p(x))의 lower bound를 구해서 최적화 시켜야만 한다.

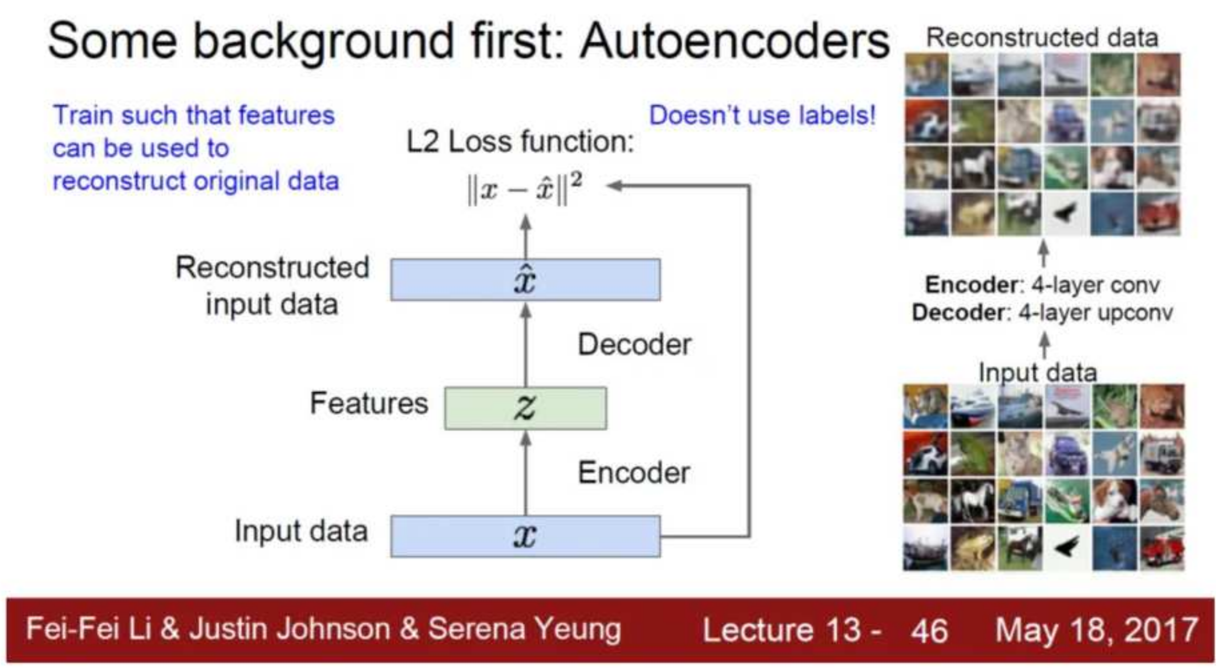
### 4.1 Autoencoders

## Some background first: Autoencoders

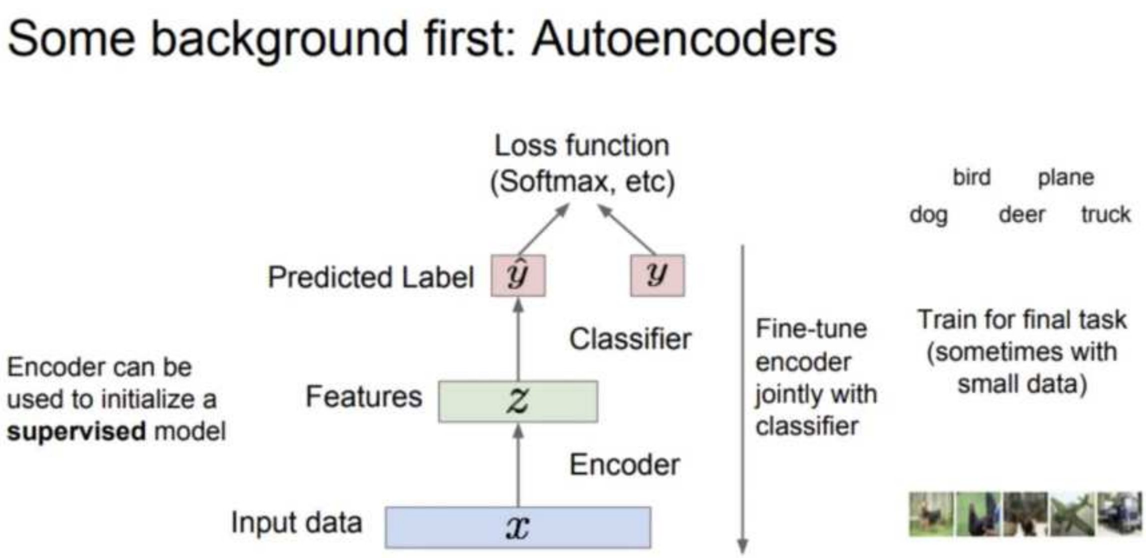
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data




- 먼저 Input image x를 latent vector z로 encoding을 해야 한다.
- z의 차원이 x 보다 작은 이유는 x의 feature를 저장하는 벡터이기 때문에 dimensionality reduction이 발생한다.



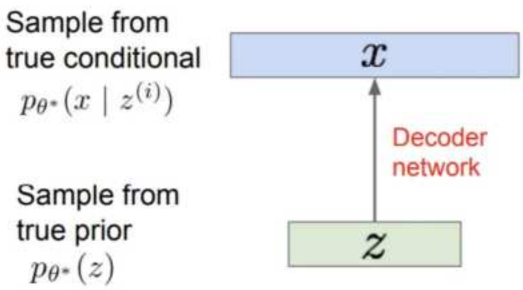

- Encoder를 통해 x → z로 변환시킨다.
- 그 후 decoder를 통해서 Reconstructed input data를 만들어 낸다.
- label를 사용하지 않고 Input data를 통해 L2 loss를 구해준다.




- train이 끝나면 decoder는 버린다.
- 위에 classifier로 붙이면 클래스 label를 출력하는 분류 문제로 바뀐다.
- Autoencoder는 label되지 않는 데이터로부터 양질의 general feature representation을 학습할 수 있는 장점이 있다.

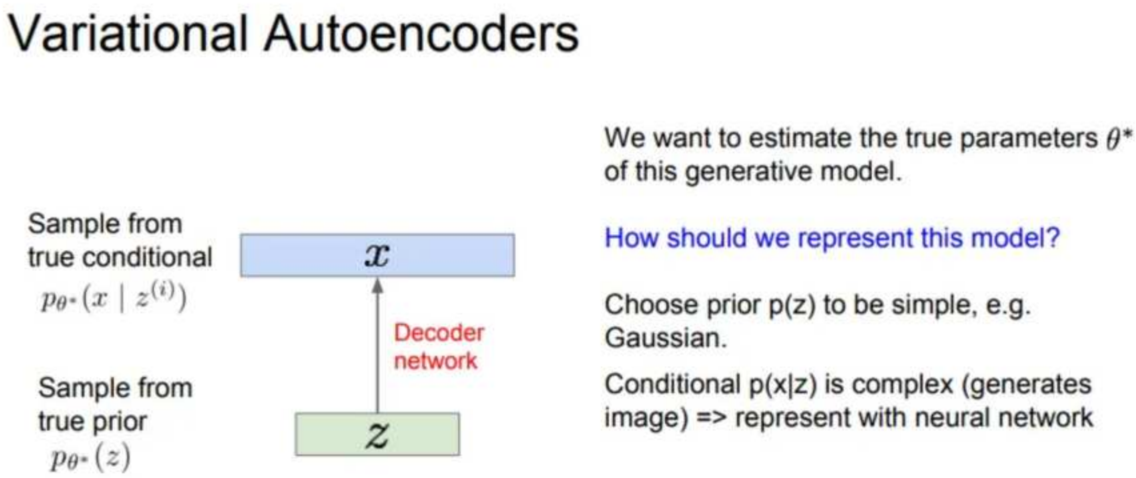
- 학습시킨 feature representation을 데이터가 부족한 지도학습 모델의 가중치로 이용할 수 있다.

#### 4.2 VAE

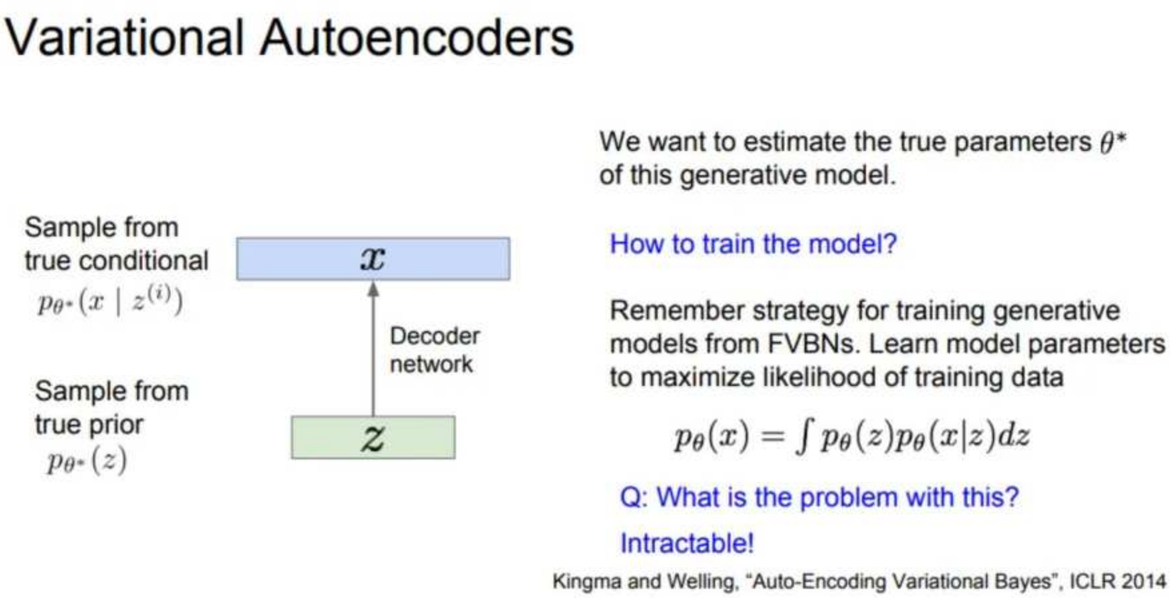



- Auto-Encoder가 잘 추출한 feature를 사용해 이미지 클래스를 분류했다면, VAE는 이 feature로 새로운 이미지를 생성할 수 없을까? 라는 질문에서 시작한다.

$$z : latent\ vector$$
$$P_\theta(z) : parameter\text{가 } \theta\text{일때}, latent\ vector\ z\text{를 } sampling \text{ 할 수 있는 확률밀도함수}$$
$$P_\theta(x|z^{(i)}) : parameter\text{가 } \theta\text{이면서}, z\text{가 주어졌을때 } x\text{를 생성하는 확률밀도함수}$$



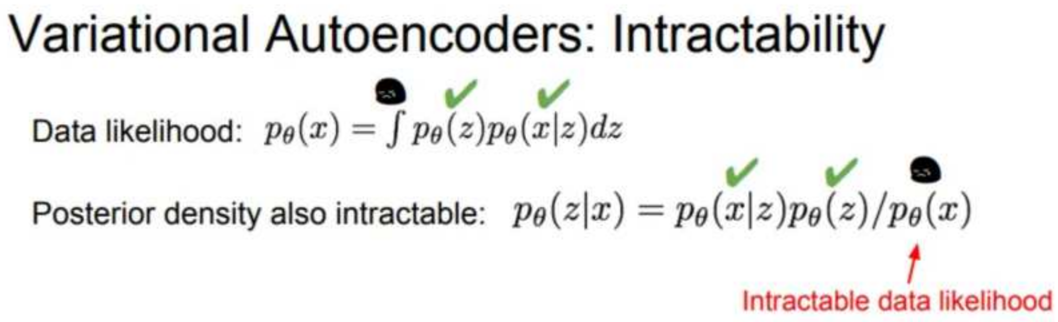

- p(z)를 보통 Gaussian으로 만든다.
- 얻은 z 분포를 가지고 p(x|z)를 샘플링한다.




- 우리가 VAE를 통해 얻고자 하는 것은 true parameter θ이다.

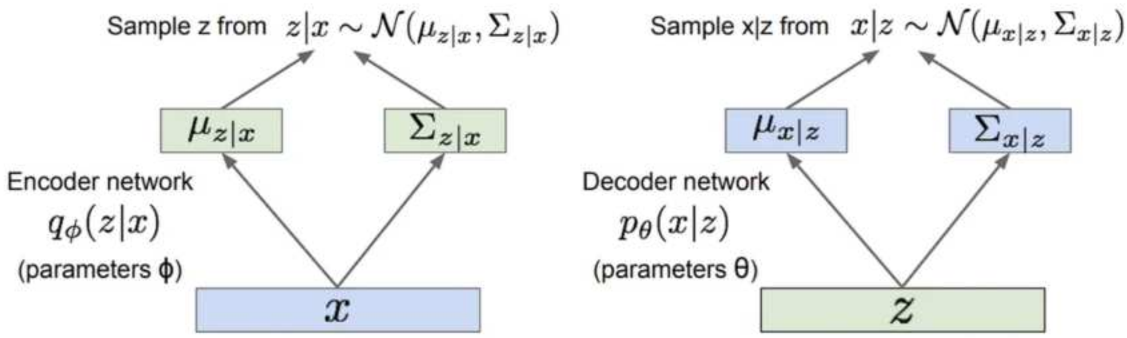
$$P_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

- **기대값 = 적분(확률변수 * 확률)**
- 우리는 FVBN(Fully Visible Brief Network)를 사용할 것이다.
- x의 likelihood를 최대화하는 확률분포를 찾는 것이다.
- 문제는 이 적분식이 Intractable하기 때문에 최적화를 직접할 수 없다.
- 따라서 lower bound를 유도하여 likelihood를 구하게 된다.
- 우리는 decoder network를 학습시키고 싶은 것이 목적이다.
- 하지만 이 조건부 확률은 intractable하므로 다른 식을 이용해야 한다.




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

- P(x|z)가 구하기 어려워 bayes' rule에 의해 posterior density P(z|x)를 이용하여 수식을 다시 표현할 수 있다.
- p(x)가 intractable하므로 p(z|x)로 intractable하다.
- 따라서 우리는 p(z|x)를 근사하는 encoder network인 q(z|x)를 새로 정의하여 data likelihood에 대한 lower bound를 얻을 수 있다.




- 위에서는 decode network만 있었는데 여기서 encoder q(z|x)를 추가하여 autoencoder와 동일한 구조로 만들어준다.

**Encoder:**

- x를 input으로 받아서 mean, covariance 추출 후 z space 상에서 분포를 생성
- z는 gaussian 분포를 따른다고 가정. (다른 분포도 가능)

**Decoder:**

- gaussian 분포로부터 z를 sampling
- sampling한 z를 가지고 decoder p(x|z)는 x space 상의 확률 분포를 생성하고, x를 이 분포로부터 sampling

# Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)} \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)} \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)} \\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}
\end{aligned}
$$

Reconstruct the input data

Make approximate posterior distribution close to prior

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

Variational lower bound ("ELBO")

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

Training: Maximize lower bound

$$
\begin{aligned}
logp_\theta(x^{(i)}) &= E_{z\sim q_\phi(z|x_i)}[logp_\theta(x^{(i)})] \ (p_\theta(x^{(i)}) \ Does\ not\ depend\ on\ z, \text{기대값} = \text{적분값}) \\
&= E_z[log\frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(x)}] \ (Bayes'Rule) \\
&= E_z[log\frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(x)}\frac{q_\phi(z|x^i)}{q_\phi(z|x^i)}] \ (Multiply\ by\ constant) \\
&= E_z[logp_\theta(x^{(i)}|z) \ - E_z[\frac{q_\phi(z|x^{(i)})}{p_\theta(z)}] + E_z[log\frac{q_\phi(z|x^i)}{p_\theta(z|x^i)}] \ (Logarithms) \\
&= E_z[logp_\theta(x^{(i)}|z) - D_{KL}(q_\phi(z|x^{(i)}||p_\theta(z)) + D_{KL}(q_\phi(z|x^{(i)}||p_\theta(z|x^{(i)}))
\end{aligned}
$$

$$
\begin{gathered}
D_{KL}(p||q) : KL\ Divergence, \text{두 } pdf\ p\text{와 } q\text{의 비슷한정도} \\
(1)\text{비슷하면 낮은 값, 서로 다른면 높은 값} \\
(2) D_{KL} \geq 0 \\
D_{KL}(P||Q) = \sum_{x\in X} P(x)log_b(\frac{P(x)}{Q(x)}) \\
= -\sum_{x\in X} P(x)log_b(Q(x)) + \sum_{x\in X} P(x)log_b(P(x)) \\
= -E_P[log_b(Q(x))] + E_P[log_b P(x))] \\
= H_P(Q) - H(P)
\end{gathered}
$$

# Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

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

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :( But we know KL divergence always >= 0.

- p_θ(x|z): Decoder network → 계산 가능
- KL 앞 부분: Encoder network + z prior → 계산 가능
- KL 뒷 부분: p_θ(z|x) - intractable → 계산 불가능, but KL divergence가 항상 0보다 크므로 lower bound를 계산할 수 있다.

# Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}\\
&= \boxed{\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\boxed{\geq 0}}
\end{aligned}
$$

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

- 따라서 앞부분을 Tractable lower bound로 하여 gradient를 구하고 optimizer를 실행한다.

# Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}\\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}
\end{aligned}
$$

Reconstruct the input data

Make approximate posterior distribution close to prior

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

Variational lower bound ("ELBO")

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

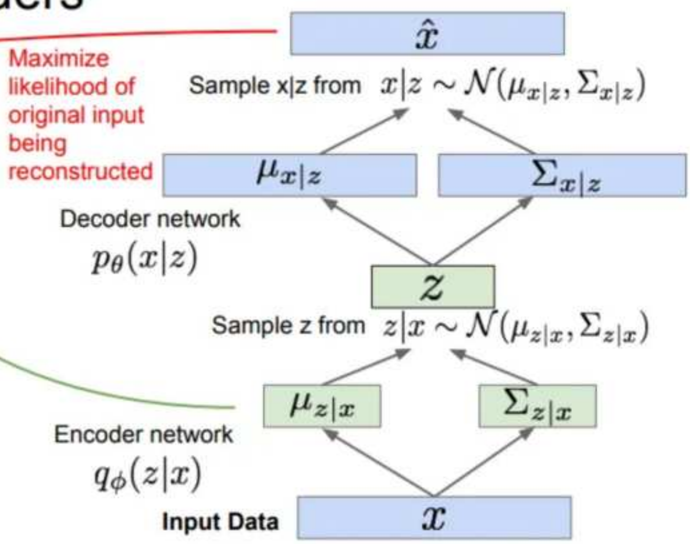
Training: Maximize lower bound

$$Variational\ lower\ bound\ ("ELBO")$$
$$log p_\theta(x^{(i)}) \geq \zeta(x^{(i)},\theta,\phi)$$
$$Training:\ Maximize\ lower\ bound$$
$$\theta^*, \phi^* = argmax_{(\theta,\phi)} \sum_{i=1}^{N} \zeta(x^{(i)},\theta,\phi)$$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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




Make approximate posterior distribution close to prior

For every minibatch of input data: compute this forward pass, and then backprop!

- Input Encoder과정에서 뒷 부분인 approximate posterior distribution을 계산한다.
- decoder부분에서 앞부분에 대한 값을 얻는다.

# Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a (variational) lower bound

**Pros:**
- Principled approach to generative models
- Allows inference of q(z|x), can be useful feature representation for other tasks

**Cons:**
- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

**Active areas of research:**
- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian
- Incorporating structure in latent variables

- VAE는 intractable density 확률 q(z|x)에 대하여 lower bound를 이용해 optimize를 하였다.
- 하지만 lower bound를 이용하기 때문에 PixelRNN/CNN보다 좋은 성능을 내지 못하고 이미지가 흐리다는 단점이 있다.

## 05_GAN(Generative Adversarial Networks)

# So far...

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

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: don't work with any explicit density function!
Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

- 지금까지는 explicit하게 density function을 구해서 직접 계산을 하였다.
- 그런데 우리가 explicit를 포기하고 그냥 sampling만 할 수 있다면 어떨까?
- 게임 이론 접근을 통해 training distribution으로부터 생성하는 방법을 학습할 것이다.

## Generative Adversarial Networks

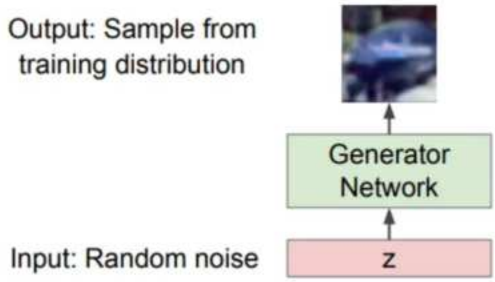
Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!



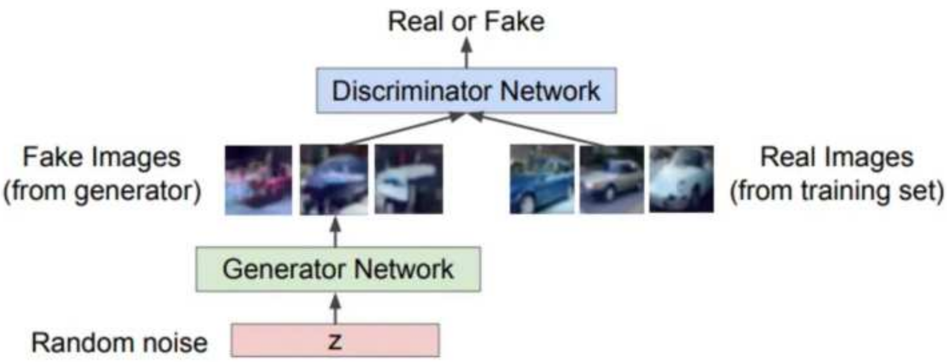

- GAN도 Neural network를 통해서 문제를 해결한다.

### 5.1 Training GANs: Two-player game

## Training GANs: Two-player game

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

**Generator network**: try to fool the discriminator by generating real-looking images
**Discriminator network**: try to distinguish between real and fake images




- 2개의 네트워크가 게임을 하는 구조이다.
- Generator network는 실제 같은 fake image를 만들어서 discriminator를 속이는 것이다.
- 반대로 Discriminator network는 실제 사진과 가짜 사진을 구별하는 것이다.

## Training GANs: Two-player game

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

**Generator network**: try to fool the discriminator by generating real-looking images
**Discriminator network**: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator ($\theta_d$) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator ($\theta_g$) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

- Discriminator는 진짜 데이터는 1로 판단하여 첫 번째 항을 0으로, 두 번째 가짜 데이터를 0으로 판단하여 두 번째 항도 0으로 만드는 것이 목표이다.
- 즉, Discriminator가 얻을 수 있는 이상적인 최대값은 0이다.
- 반대로 Generator는 두 번째 항의 D(G(z)) = 1로 판단하게 하여 두 번째 항 전체를 -∞ 만드는 것이 목표이다.
- 즉, Generator가 얻을 수 있는 이상적인 최솟값은 -∞이다.

$$Minimax\ objective\ function:$$
$$min_{\theta_g} max_{\theta_d} [E_{x \sim p_{data}} log D_{\theta_d}(x) + E_{z \sim p_z} log(1 - D_{\theta_d}(G_{\theta_g}(x)))]$$

# Training GANs: Two-player game

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

Minimax objective function:

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

Alternate between:

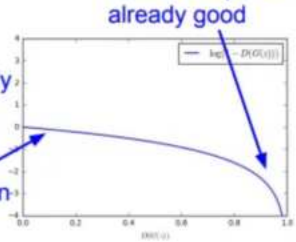
1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

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

In practice, optimizing this generator objective does not work well!



When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good

$$Gradient\ ascent\ on\ discriminator:$$
$$max_{\theta_d} [E_{x \sim p_{data}} log D_{\theta_d}(x) + E_{z \sim p_z} log(1 - D_{\theta_d}(G_{\theta_g}(x)))]$$

$$Gradient\ descent\ on\ discriminator:$$
$$min_{\theta_g} [E_{z \sim p_z} log(1 - D_{\theta_d}(G_{\theta_g}(x)))]$$

- 최소화와 최대화 문제가 같이 얽혀있기 때문에 Gradient ascent와 gradient descent를 번갈아 진행한다.
- 하지만 이렇게 하면 좋은 결과를 얻기 힘들다고 한다.
- 그래프를 한번 보자. x축은 D(G(z))를 뜻한다.
- generator의 성능은 D(G(z))값이 높을 수록 좋다.
- D(G(z))값이 작다는 것인 generator를 더 개선해줘야 하는데 gradient가 flat하기 때문에 학습에 어려움이 생긴다.
- 반대로 D(G(z))값이 너무 커지게 되면 이미 잘 완성되었지만 gradient값이 커져 문제가 발생한다.

# Training GANs: Two-player game

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

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

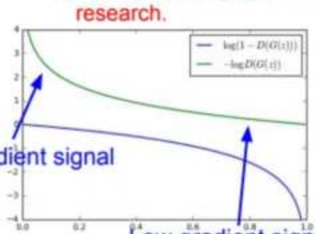
$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

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

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.




- 따라서 Generator도 Gradient ascent를 사용하여 문제를 해결한다.

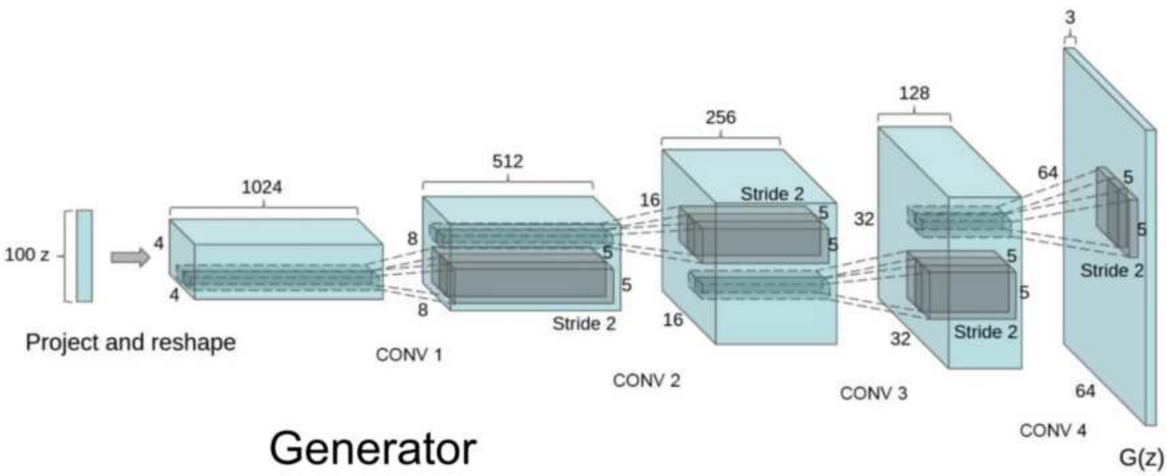
$$Gradient\ ascent\ on\ discriminator:$$
$$max_{\theta_g}[E_{z \sim p_z} log(D_{\theta_d}(G_{\theta_g}(x)))]$$

- 결과적으로 generator를 통해서 이미지를 생성할 수 있다.
- 기본적으로 GAN만 구현하면 성능이 떨어질 수 있다.

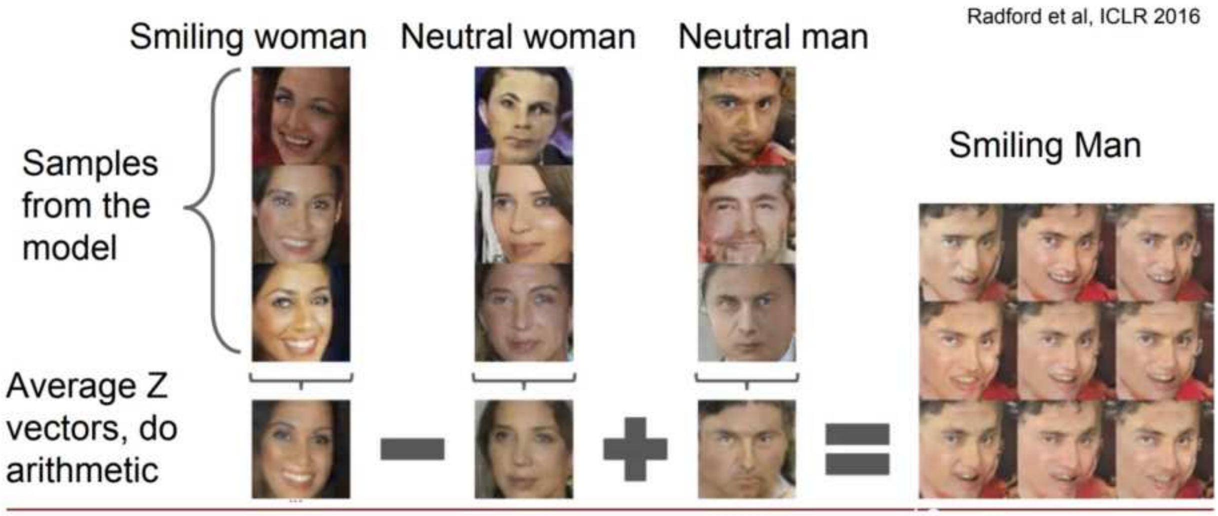
Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.




- CNN을 이용하여 이미지를 처리하고 Generator에 집어넣으면 성능이 더 개선된다.




- 이미지들의 평균을 덧셈 뺄셈을 통해서 어떤 특징을 빼고 더할 수 있는데 이것이 z의 역할이다.

Pros:
- Beautiful, state-of-the-art samples!

Cons:
- Trickier / more unstable to train
- Can't solve inference queries such as p(x), p(z|x)

- 아름다운 방법이지만 train할 때 불안정하다는 단점이 있다.