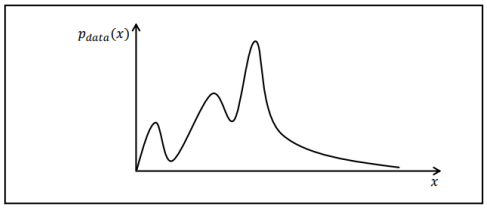
**Generative Model & VAE**

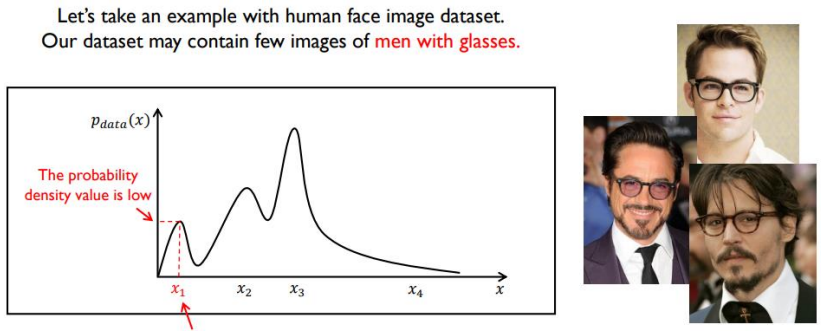
Generative Model(생성 모델)이란 그럴듯한 이미지를 생성하는 모델을 의미한다. 여기서 '그럴듯한 이미지'는 학습 데이터에 있는 이미지와 유사한 이미지라고 생각하면 된다. 예를 들어, 사람 얼굴 이미지 학습 데이터가 있을 때 생성 모델은 사람 얼굴과 유사한 이미지를 생성해야 좋은 생성 모델이다.

따라서 생성 모델의 목적은 모델이 여러 이미지들을 생성할 때, 그 이미지들의 확률 분포가 학습 데이터의 확률 분포와 닮는 것을 원한다.

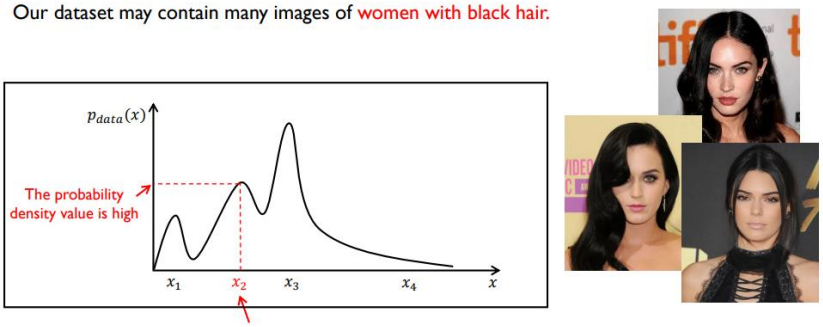
예를 들어 학습 데이터의 확률 분포($P_{data}(x)$)가 다음과 같이 생겼다고 가정하자. x축은 (특정한 이미지들의 그룹)이고, y축은 (해당 그룹에 속하는 이미지가 추출될 확률)이다.




간단한 예시들을 들어보겠다. x1은 '안경 낀 남자 얼굴 이미지'에 대한 그룹이다. 그런데 만약 학습 데이터에 이런 이미지들이 적다면 이런 이미지들을 추출할 확률은 적을 것이다. 따라서 y값인 $P_{data}(x)$는 작은 값을 지닌다.

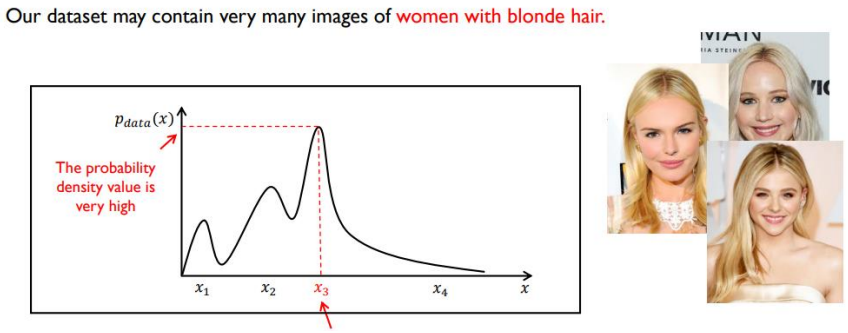



x2는 '검은색 머리를 지닌 여자 얼굴 이미지'에 대한 그룹이다. 이런 이미지들은 x1의 이미지들보다 더 많다면 $P_{data}(x)$는 좀 더 큰 값을 지닌다.



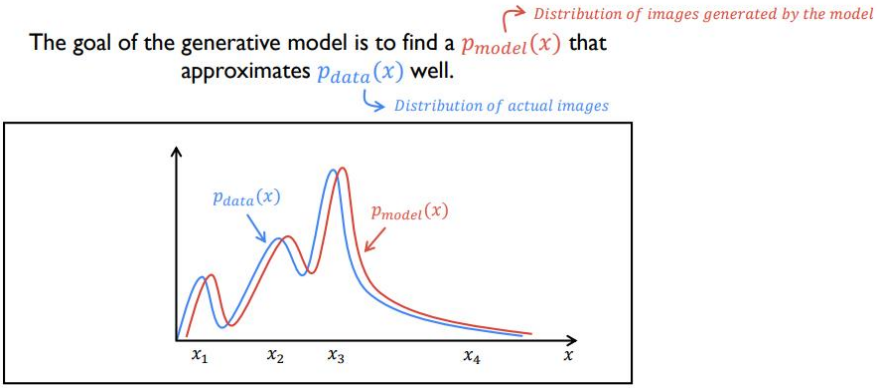

x3는 '금발 머리를 지닌 여자 얼굴 이미지'에 대한 그룹이다. 만약 이러한 이미지들이 학습 데이

터에 가장 많다면 추출될 확률($P_{data}(x)$)도 가장 높을 것이다.




그렇다면 $P_{data}(x)$가 0인 이미지들은 어떤 이미지들일까? 예상해보자면 사람 얼굴과는 전혀 관계가 없는 동물이나 사물, 풍경 등의 이미지를 나타낼 것이다.

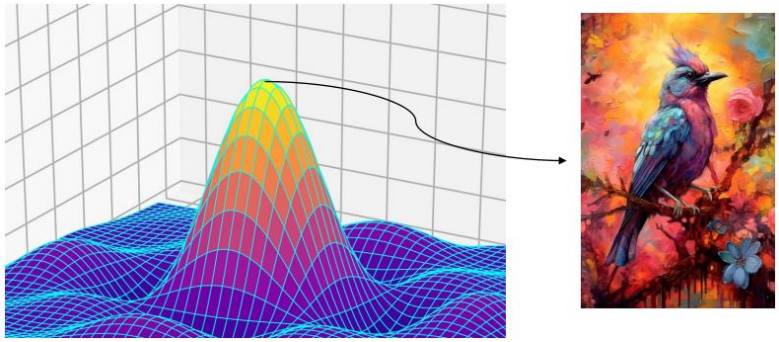
이제 다시 생성 모델의 목적에 집중해보자.
모델에 의해 생성된 이미지들의 확률 분포를 $P_{model}(x)$라고 하고, 실제 이미지들의 확률 분포를 $P_{data}(x)$라고 할 때 두 분포의 차이가 적기를 원하는 것이다.




다시 예를 들면, 생성 모델이 이미지를 생성할 때 '안경 낀 남자 얼굴 이미지'를 생성할 확률($P_{model}(x_1)$)이 실제 데이터에서 '안경 낀 남자 얼굴 이미지'를 추출할 확률($P_{data}(x_1)$)이 되기를 원한다는 것이다. 수치적으로 예시를 들면, 실제 데이터가 100개의 얼굴 이미지이고, x1그룹의 이미지를 뽑을 확률이 1/10이라고 하자. 그럼 생성 모델도 x1그룹의 이미지를 생성할 확률이 1/10이 되기를 원하는 것이다.

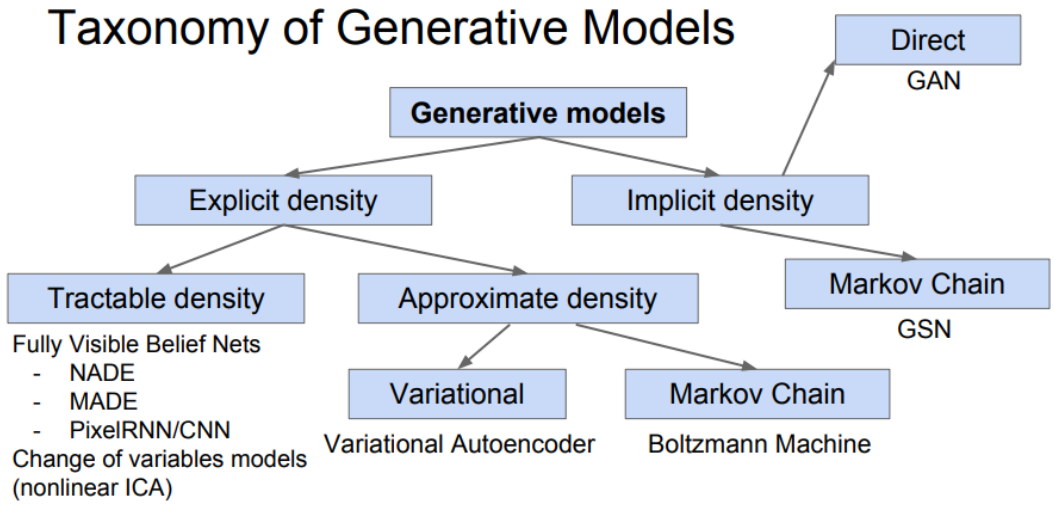
즉 실제 데이터의 분포에 맞게 얼굴 이미지들을 생성하고, 관련이 없는 동물, 사물, 배경 등은 전혀 생성하지 않는 모델을 만들겠다고 이해하면 된다.

생성 모델의 학습이 완료되었으면 $P_{model}(x)$ 분포에서 Sampling(생성)을 할 수 있다. 샘플링은 랜덤한 z 벡터를 생성 모델에 입력하여 이미지를 생성하는 과정으로 구현한다. 왜 이렇게 구현되는지는 VAE에 대한 설명을 끝까지 보면 이해할 수 있을 것이다.



이러한 생성 모델을 만드는 방법은 여러 가지가 있다.

생성 모델의 분류는 크게 Explicit density와 Implicit density로 나뉘어지고, Explicit density의 대표 모델로는 VAE, Implicit density의 대표 모델로는 GAN이 있다.




두 분류 모두 생성 모델에 속하기 때문에, 모델이 훈련 데이터에 있는 이미지를 생성하는 것을 원하고, 결국 그렇게 만들어진 이미지들의 분포가 실제 데이터의 분포와 유사해지기를 원한다.

이것을 수학적으로 표현하면 [p(x): 훈련 데이터에 있는 이미지 x를 generator가 생성할 가능성]를 높여서 결국 $P_{model}(x)$가 $P_{data}(x)$가 되기를 원한다.

그리고 Explicit과 Implicit의 차이는 p(x)를 정의하여 이미지를 생성할 것인지, p(x)를 정의하지 않고 이미지를 생성할 것인지를 나타낸다. 본 포스팅은 Explicit 모델을 다룰 것이다.

다루기 쉬운(tractable한) p(x)같은 경우, 아래와 같은 확률밀도함수(PDF)로 정의할 수 있다.

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

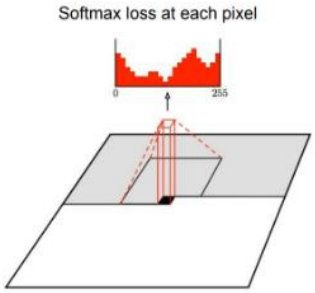
Likelihood of image x

Probability of i'th pixel value given all previous pixels

이전 픽셀($x_1, x_2, .., x_{i-1}$)들이 주어졌을 때 현재 픽셀($x_i$)을 정답 픽셀로 잘 예측할 확률에 대한 곱으로

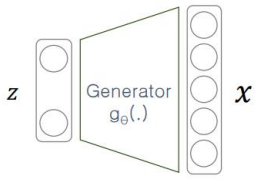
이미지가 생성될 가능성(p(x))을 정의하는 것이다.

이렇게 정의하고 p(x)를 Maximize하는 접근법으로 이미지를 생성할 수 있다. 즉 Maximum likelihood 문제를 풀면 이미지를 생성할 수 있다는 논리이며, 이것을 잘 구현한 모델이 PixelCNN 이다.




PixelCNN은 Convolution kernel을 이용해서 이전 픽셀들이 주어졌을 때 현재 픽셀에 대한 분류를 진행한다. 이 분류된 값이 정답일 확률이 maximize되도록 학습을 진행한다. 그러나 이러한 접근법은 코너에서부터 순차적으로 픽셀들을 생성해야 하므로 효율적이지 못하다는 단점이 있다.

따라서 Intractable한 p(x)를 설정하고, p(x)를 근사하는 모델들이 대두되었다. 그 중 하나가 바로 VAE이다. VAE의 저자들은 간단한 표준 가우시안 분포 p(z)로부터 z를 샘플링하고, 디코더에 z를 입력하여 이미지 x를 생성하려고 했다.




| | |
|---|---|
| $z \sim p(z)$ | Random variable |
| $g_\theta(\cdot)$ | Deterministic function parameterized by θ |
| $x = g_\theta(z)$ | Random variable |

$$p(x|g_\theta(z)) = p_\theta(x|z)$$

그리고 p(x)를 z에 대한 PDF로 정의하였다. (전체 확률의 법칙 이용)

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

하지만 모든 z에 대해 p(x|z)를 계산할 수 없었기 때문에 p(x)가 Intractable하였다. 따라서 p(x)의 분포를 알 수 없었고, MLE도 진행하지 못하였다. 또한 사후확률(posterior)인 p(z|x)또한 계산할 수 없었다. (베이지안룰 이용)

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

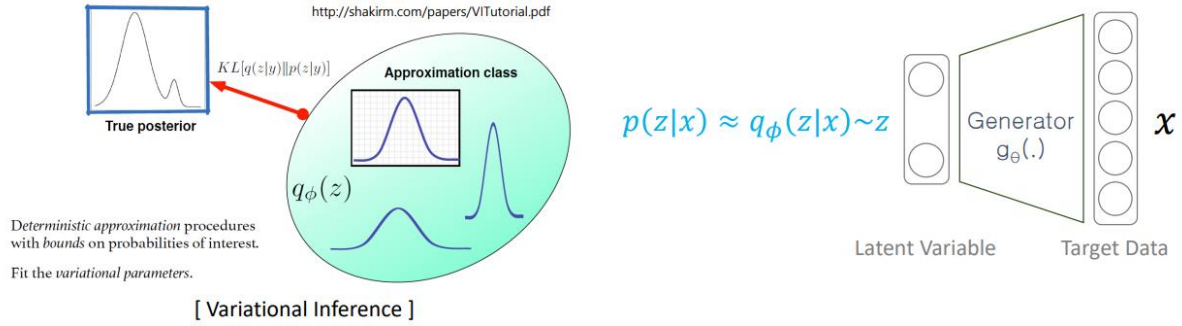
만약 posterior(p(z|x))를 알았다면 위 수식처럼 베이지안룰로 p(x)도 알 수 있을 것이다. 이처럼 p(z|x)는 p(x)를 알기 위한 키 역할을 수행할 수 있으며, 이 분포의 의미는 어떤 이미지 x가 이상적인 feature extractor에 들어왔을 때 z에 대한 분포(x를 아주 잘 설명하는 feature 분포)라고 생각할 수 있다고 한다.

하지만 p(z|x)는 단순하게 알 수 없기 때문에, 저자들은 한 가지 아이디어를 내는데, 바로 p(z|x)를 근사하는 q(z|x)를 Neural Network (Encoder)로 제작하는 것이었다.

이 때 q(z|x)는 우리가 알고 있는 확률분포(ex. 가우시안) 중 하나를 택하고, 그 분포의 파라미터값(ex. 평균, 표준편차)을 조정하여 실제 분포인 p(z|x)와 유사하게 만들려고 하였다. 다시 말해, p(z|x) 분포와 유사한 가우시안 분포, q(z|x)를 Neural Network로 찾겠다는 것이다.

P.s. Neural Network를 통해 가우시안 분포를 만들겠다는 의미는 NN의 출력을 평균과 표준편차로 만들고, 해당 파라미터(평균, 표준편차)를 따르는 가우시안 분포가 있다고 생각하는 것이다.

이것이 바로, 알고 싶은 분포(p(z|x))를 알기 쉬운 분포(q(z|x), gaussian)로 근사하는 Variational Inference(변분 추론)라고 한다.



[ Variational Inference ]

학습을 통해 q(z|x)가 p(z|x)와 정말 유사해졌다면, q(z|x)에서 샘플링한 z를 디코더에 입력하면 그럴듯한 x를 만들 수 있을 것이다.

다음은 학습 방법에 대한 설명이다.

Variational Inference를 하기 위해서는 q(z|x)와 p(z|x)의 KL Divergence(두 확률분포에 대한 거리)가 낮도록 학습해 실제 p(z|x)와 근사한 가우시안 분포를 찾을 필요성이 있었다. 하지만 p(z|x)는 intractable하므로 바로 KLD를 풀 수가 없었다.

따라서 저자들은 p(x)에 대한 PDF를 ELBO와 KL term의 합으로 정리하였고, ELBO를 maximization 하면서 자연스럽게 KL를 minimization하는 방법을 이용했다.

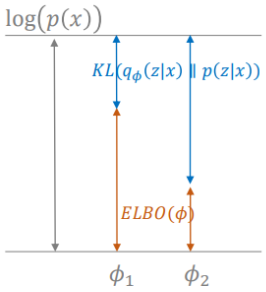
$$\log(p(x)) = \int \log(p(x)) q_\phi(z|x) dz \quad \leftarrow \int q_\phi(z|x)dz = 1$$

$$= \int \log\left(\frac{p(x,z)}{p(z|x)}\right) q_\phi(z|x) dz \quad \leftarrow p(x) = \frac{p(x,z)}{p(z|x)}$$

$$= \int \log\left(\frac{p(x,z)}{q_\phi(z|x)} \cdot \frac{q_\phi(z|x)}{p(z|x)}\right) q_\phi(z|x) dz$$

$$= \underbrace{\int \log\left(\frac{p(x,z)}{q_\phi(z|x)}\right) q_\phi(z|x) dz}_{ELBO(\phi)} + \underbrace{\int \log\left(\frac{q_\phi(z|x)}{p(z|x)}\right) q_\phi(z|x) dz}_{KL\left(q_\phi(z|x) \,\|\, p(z|x)\right)}$$

두 확률분포 간의 거리≥0




여기서 ELBO란, Evidence LowerBOund로, 하한을 의미한다. 즉 log(p(x))>=ELBO를 의미하고, 이것이 가능한 이유는 KL term이 항상 0이상을 만족하기 때문이다.
또한 ELBO를 최대화하겠다는 것은 log(p(x))도 같이 최적화한다는 의미이다. p(x)가 intractable하므로 직접 최적화는 못하지만 ELBO를 최적화해서 p(x)도 같이 올리는 전략이라고 볼 수 있다.

ELBO를 다시 정리해보니 다음과 같이 log term과 KL term으로 전개됐다. (조건부 기댓값 이용)

$$ELBO(\phi) = \int \log\left(\frac{p(x,z)}{q_\phi(z|x)}\right) q_\phi(z|x) dz$$

$$= \int \log\left(\frac{p(x|z)p(z)}{q_\phi(z|x)}\right) q_\phi(z|x) dz$$

$$= \int \log(p(x|z)) q_\phi(z|x) dz - \int \log\left(\frac{q_\phi(z|x)}{p(z)}\right) q_\phi(z|x) dz$$

$$= \mathbb{E}_{q_\phi(z|x)}[\log(p(x|z))] - KL(q_\phi(z|x)||p(z))$$ 앞 슬라이드에서의 KL과 인자가 다른 것에 유의

ELBO를 maximization한다는 의미는 negative ELBO를 minimize한다는 의미와 같으므로 위 수식에 minus를 붙여서 목적함수는 아래와 같이 표현할 수 있다.

$$\arg\min_{\phi,\theta} \sum_i \underbrace{-\mathbb{E}_{q_\phi(z|x_i)}[\log(p(x_i|g_\theta(z)))] + KL(q_\phi(z|x_i)||p(z))}_{L_i(\phi,\theta,x_i)}$$

이 목적함수를 분석하면 Reconstruction Error term과 Regualarization term이 나왔다고 한다.

$$L_i(\phi,\theta,x_i) = \underbrace{-\mathbb{E}_{q_\phi(z|x_i)}[\log(p(x_i|g_\theta(z)))]}_{\textbf{Reconstruction Error}} + \underbrace{KL(q_\phi(z|x_i)||p(z))}_{\textbf{Regularization}}$$

먼저 Reconstruction Error는 (-log를 지우고 생각하면) $x_i$(정답 픽셀)에 대한 MLE로 생각하면 되는데, g(z)가 $x_i$를 예측한 값을 평균으로 지니는 정규분포를 출력하면 해당 분포에서 실제 $x_i$의 가능성을 최대로 맞추도록 최적화 해야한다는 의미이다. (실제 $x_i$의 값을 평균으로 지니도록 최적화 되

어야 함) 수식에 E(기댓값)가 붙은 이유는 모든 픽셀에 대한 MLE를 높여야 하기 때문이다.

Regularization은 q(z|x)의 분포가 p(z)의 분포와 유사해야 한다는 조건이며, 이 조건을 만족해야 ELBO가 Maximization되었을 때, KL(p(z|x) || q(z|x))을 Minimization 된다는 것을 의미한다. 이 때 q(z|x)와 p(z)는 모두 가우시안을 따르므로 다음과 같이 계산하기 쉬운 식 형태로 변한다고 한다.

$$KL(q_\phi(z|x_i)||p(z)) = \frac{1}{2}\left\{tr(\sigma_i^2 I) + \mu_i^T \mu_i - J + \ln\frac{1}{\prod_{j=1}^{J}\sigma_{i,j}^2}\right\}$$

$$= \frac{1}{2}\left\{\sum_{j=1}^{J}\sigma_{i,j}^2 + \sum_{j=1}^{J}\mu_{i,j}^2 - J - \sum_{j=1}^{J}\ln(\sigma_{i,j}^2)\right\}$$

$$= \frac{1}{2}\sum_{j=1}^{J}(\mu_{i,j}^2 + \sigma_{i,j}^2 - \ln(\sigma_{i,j}^2) - 1) \quad \text{Easy to compute!!}$$

**Kullback–Leibler divergence** [edit]

The Kullback–Leibler divergence from $\mathcal{N}_0(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ to $\mathcal{N}_1(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$, for non-singular matrices $\Sigma_0$ and $\Sigma_1$, is:[8]

$$D_{\mathrm{KL}}(\mathcal{N}_0\|\mathcal{N}_1) = \frac{1}{2}\left\{\operatorname{tr}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0\right) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^{\mathrm{T}}\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - k + \ln\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|}\right\},$$

(posterior, prior)

where $k$ is the dimension of the vector space.

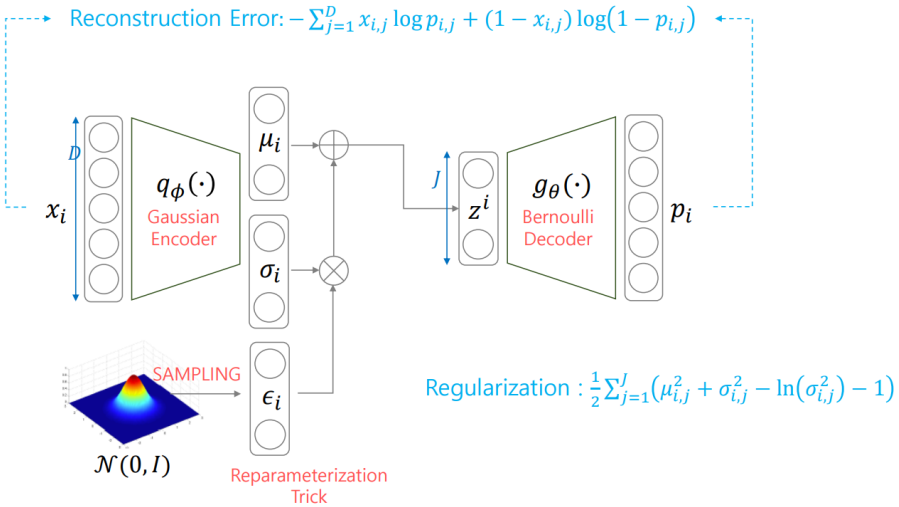
결국 Regularization term은 두 가지 효과가 있다. 이 효과는 매우 중요하니 숙지하는 것을 권장한다.

1) q(z|x)와 p(z|x)를 유사하게 만들어서 q(z|x)에서 샘플링한 z가 디코더에 들어가면 x를 잘 reconstruction하도록 한다.

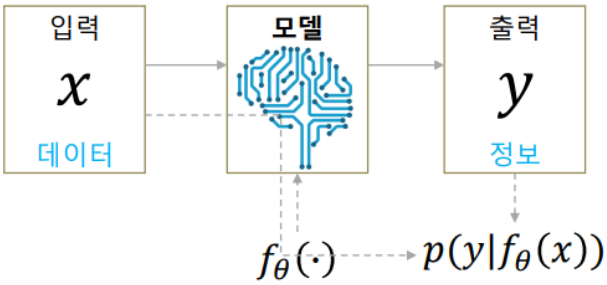
2) q(z|x)와 p(z)의 분포가 유사해졌다면, 생성을 할 때, 표준 가우시안 분포(p(z))에서 샘플링한 z가 q(z|x)의 분포에 속할 수 있다. 따라서 인코더에서 샘플링한 것과 비슷한 효과를 준다.

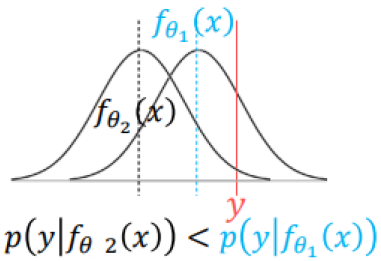
P.s. 다만 실제로는 인코더에서 샘플링한 것은 q(z|x)의 평균과 가까울텐데, p(z)에서 샘플링한 것이 q(z|x)의 평균과 가깝다는 것을 보장하지 못하므로 생성 능력이 그렇게 좋지는 않다.

지금까지 설명한 내용은 아래의 그림으로 정리할 수 있다.



**Maximum Likelihood Estimation**




$p(y|f_{\theta\ 2}(x)) < p(y|f_{\theta_1}(x))$

Likelihood: PDF의 y값으로, p(y|f(x)의 y축 값으로 생각하면 된다.

MLE: 정해진 확률분포에서 출력이 나올 확률을 최대로 나오는 방법
정해진 확률분포(p(y|f(x))에서 출력(y)가 나올 확률을 최대로 만들겠다. 그림에서 $f_{theta2}(x)$였던 학습 모델을 업데이트해서 $f_{theta1}(x)$으로 만들면서 점점 y에 대한 확률을 키우겠다고 이해할 수 있다.
-> 모델이 생성한 확률분포에서 정답에 대한 likelihood를 최대로 만들겠다. (그 방법으로 모델을 Gradient Descent로 업데이트하는 것임)

MLE를 푸는 것은 negative log likelihood를 최소화시키는 것과 같다. 이 때 f(x)의 분포가 가우시안이면 MSE를 낮추면 되고, 베르누이면 Cross Entropy를 낮추면 된다.

**Univariate cases**

$$-\log(p(y_i|f_\theta(x_i)))$$

**Gaussian distribution**

$$f_\theta(x_i) = \mu_i, \sigma_i = 1$$

$$p(y_i|\mu_i,\sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{(y_i-\mu_i)^2}{2\sigma_i^2}\right)$$

$$\log(p(y_i|\mu_i,\sigma_i)) = \log\frac{1}{\sqrt{2\pi}\sigma_i} - \frac{(y_i-\mu_i)^2}{2\sigma_i^2}$$

$$-\log(p(y_i|\mu_i)) = -\log\frac{1}{\sqrt{2\pi}} + \frac{(y_i-\mu_i)^2}{2}$$

$$-\log(p(y_i|\mu_i)) \propto \frac{(y_i-\mu_i)^2}{2} = \frac{(y_i-f_\theta(x_i))^2}{2}$$

Mean Squared Error

**Bernoulli distribution**

$$f_\theta(x_i) = p_i$$

$$p(y_i|p_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

$$\log(p(y_i|p_i)) = y_i\log p_i + (1-y_i)\log(1-p_i)$$

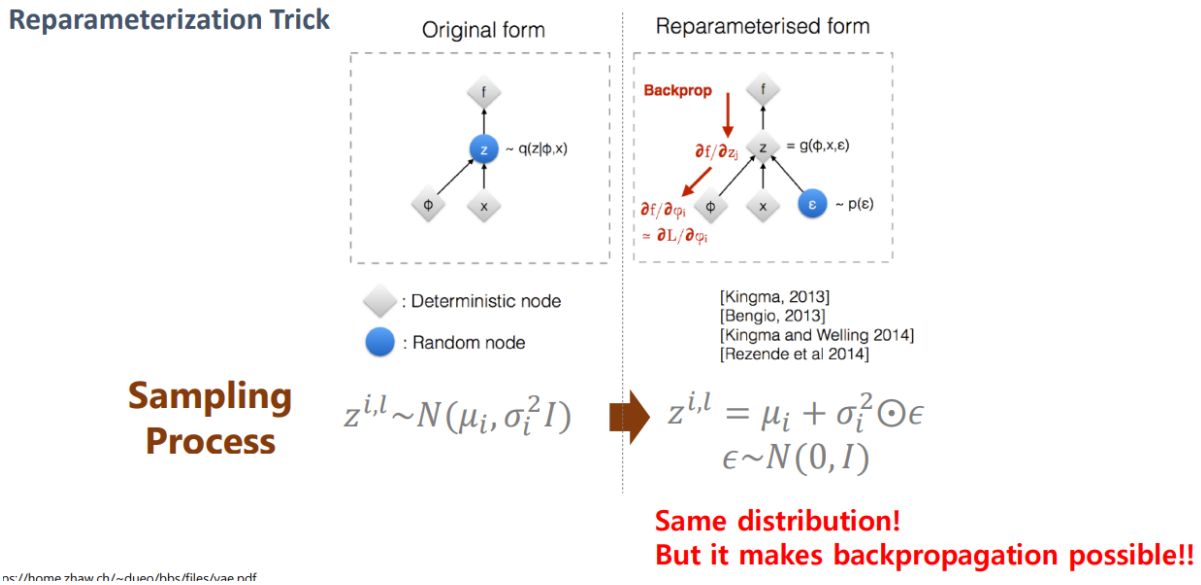
$$-\log(p(y_i|p_i)) = -[y_i\log p_i + (1-y_i)\log(1-p_i)]$$

Cross-entropy

**reparameterization trick**

q(z|x)의 분포에서 z를 추출하기 위해서는 랜덤 함수를 이용해야 하는데, 랜덤 함수에 대한 역전파

가 존재하지 않으므로 학습이 되지 않는다. 따라서 trick을 이용하는데, 그 방법은 정규분포를 표준정규분포로 바꾸는 수식을 역으로 이용하는 것이다. [z=(x-mu)/sigma, z는 표준 정규 분포의 값, x는 정규 분포의 값]이라고 할 때, [x=z*sigma+mu]로 계산할 수 있다. 이 때 z를 [N(0,1): 표준정규분포]에서 샘플링한 값(epsilon)으로 생각하고, x를 z라는 새로운 변수로 치환하면, z=mu+sigma*epsilon으로 바꿔 쓸 수 있다. 그래서 결론은 표준정규분포에서 샘플링한 값을 이용해서 정규분포에서 샘플링한 값을 계산하는 것이다. 이렇게 계산을 하면 epsilon은 forward과정에서 저장할 수 있고, 샘플링을 epsilon, mu, sigma를 가지고 곱셈과 덧셈을 통해 계산하므로 backward도 수행할 수 있다.




**VAE 구현**

encoder를 통해 나온 z feature(크기가 2)를 통해 크기가 2인 mu, sigma 벡터들을 각각 만든다. 이때 sigma는 ReLU를 거치고 0.05따위를 더해 0이상인 값이 되도록 한다. 왜냐하면 sigma는 수학적으로 0이상이며, 0일 때는 peak한 현상(한 값만 엄청 높은 분포)을 보이므로 실제의 분포와 괴리감이 있기 때문이다.

1. reconstruction loss (BCE)

: MNIST에서 BCE구하면 batchx32x32개의 loss값이 나온다. 이것을 x축으로 더하고 y축으로 더한 다음 batch크기만큼 평균하여 계산한다..

```
recon_loss = tf.keras.losses.binary_crossentropy(img_list[batch_num],out)
recon_loss = tf.reduce_sum(recon_loss, axis=(1,2))
recon_loss = tf.reduce_mean(recon_loss)
```

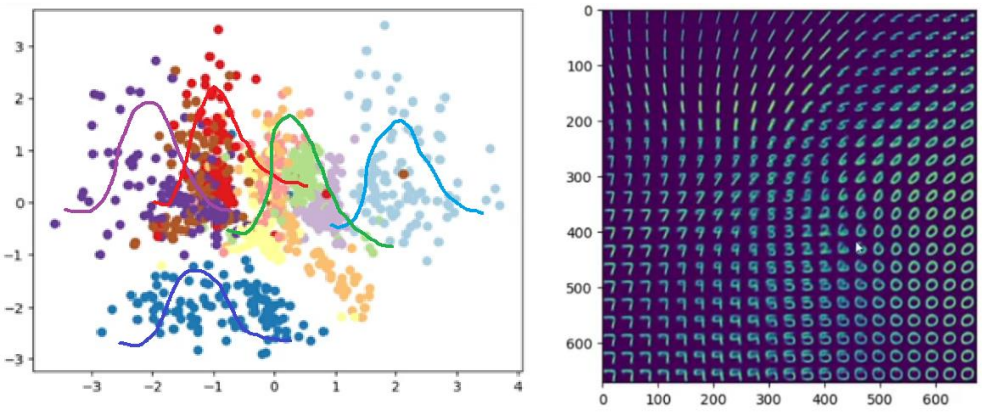
2. regularization loss (kl loss)

: KL loss를 구하면 feature 분포의 개수만큼 loss값이 나온다. 이것을 한 축으로 더한 다음 batch

크기만큼 평균하여 계산한다.

```
reg_loss = -0.5 * (1 + tf.math.log(tf.square(sigma)) - tf.square(mu) - tf.square(sigma))
reg_loss = tf.reduce_sum(reg_loss, axis=1)
reg_loss = tf.reduce_mean(reg_loss)
```

학습 이후 샘플링된 z를 matplotlib으로 시각화하면 다음과 같다. AE와 다른 점은 각각의 feature 들이 확률분포로부터 샘플링되었다는 것이다. (이 때, 비슷한 클래스의 이미지에 대해서는 잠재 변수의 확률분포가 유사할 수 있음) 이것은 디코더를 통해 이미지를 생성할 때 확률적으로 생성한다는 의미를 지닌다. (그림판의 한계로 1D 분포로 그리지만 실제로는 2D 분포이다.)




이것이 왜 의미있는가를 알려면 디코더를 통해 생성해보면 알 수 있다.
(x축은 (-10~5), y축은 (-5~5)사이를 각각 21개로 나누어서 21x21개의 값을 임의로 만들고, 그 값을 바탕으로 디코더를 통해 생성함)
결과를 확인하면 AE보다 이미지가 좀 더 그럴듯하게 생성된다고 한다. 이유는 feature가 어떤 확률 분포(1클래스에 대한 확률 분포, 0클래스에 대한 확률 분포 등)에 속하기만 하면, 그 feature를 통해 reconstruction(1로 재구축, 2로 재구축 등)이 되도록 학습을 하였기 때문에, 테스트 시 랜덤 feature가 입력되더라도 임의의 확률 분포에 속하기만 한다면, 해당 클래스의 이미지로 생성이 가능하다. 다시 말해, 랜덤 feature가 학습된 특정 값이 아니더라도 생성이 가능하도록 설계된 구조이다.  이렇게 랜덤 값으로부터 이미지를 생성하는 것이 곧 [$P_{model}(x)$: 생성된 이미지들의 확률 분포]에서 이미지를 샘플링하는 것으로 생각할 수 있다.

VAE의 특징으로는 같은 클래스라고 하더라도 여러 확률 분포가 나타날 수 있다. 예를 들어 사람에 대한 확률 분포가 있고, 그 확률 분포가 존재하는 space는 x축의 값에 따라 코의 높낮이가 결정된다고 하자. 이 때 백인 이미지가 들어오면 x축이 큰 쪽에 평균이 위치한 정규 분포가 나올 수 있다. (해당 분포해서 샘플링해서 학습됨) 그렇지 않고 황인 이미지가 들어오면 x축이 작은 쪽에 평균이 위치한 정규 분포가 나올 수 있다. 그럼 테스트 시에 어떤 x값이 입력됨에 따라 코가 작은 사람이 생성될 수도 있고, 코가 큰 사람이 생성될 수도 있다.

**VAE 요약 (이론 관점)**

Variational Inference를 이용하는 오토인코더이다. feature learning이 목적인 오토인코더와 달리 훈련 데이터의 분포를 학습 후 샘플링을 하는 생성 모델로써 explicit density 분류에 속한다. 따라서 실제 이미지를 생성할 확률인 p(x)를 사전에 정의해서 MLE를 풀려고 했으나 p(x)의 pdf와 posterior인 p(z|x) 모두 intractable해서 계산할 수가 없었다. 따라서 p(z|x)를 근사하는 encoder, q(z|x)를 도입하였고, q(z|x)는 알기 쉬운 가우시안 분포로 선택하여 variational inference를 하였다. variational inference를 하기 위해서는 KLD를 낮출 필요성이 있었는데, 저자들은 p(x)를 ELBO+KL term으로 정리하여, ELBO를 maximization함으로써 자연스레 KL을 낮추는 전략을 취하였다. 또한 ELBO가 최대화됨으로써 p(x) 또한 최적화되어 실제 이미지를 잘 생성할 수 있었다.

**VAE 요약 (구현 관점)**

오토인코더에 이미지를 생성하는 기능까지 추가한 모델이라고 생각할 수 있다. 오토인코더는 이미지를 재구축하는 학습을 하면서 입력 이미지를 잘 표현하는 latent vector, z를 제작할 수 있다. 그런데 z는 scalar값이 모인 벡터이므로, 디코더(Generator) 입장에서는 학습된 값이 들어왔을 때만 학습 데이터의 이미지와 유사하게 생성할 수 있다. 하지만 좋은 생성 모델은 랜덤한 z가 Generator의 입력으로 오더라도, 이미지를 그럴듯하게 생성해야 한다. (생성 모델의 output 분포에서 랜덤 샘플링했을 때 그럴듯한 이미지가 나와야 실제 데이터의 분포에 근사하다고 생각할 수 있기 때문) 따라서 VAE는 Encoder가 확률분포를 생성하면 해당 분포에서 샘플링한 벡터를 z로 간주한다. 이후 z를 Decoder에 입력하여 이미지를 reconstruction하도록 학습한다. 이렇게 학습하면, 인퍼런스 과정에서 랜덤한 z가 입력된다고 하더라도 학습했던 분포에 속할 확률이 높기 때문에 이미지를 그럴듯하게 생성할 수 있다.