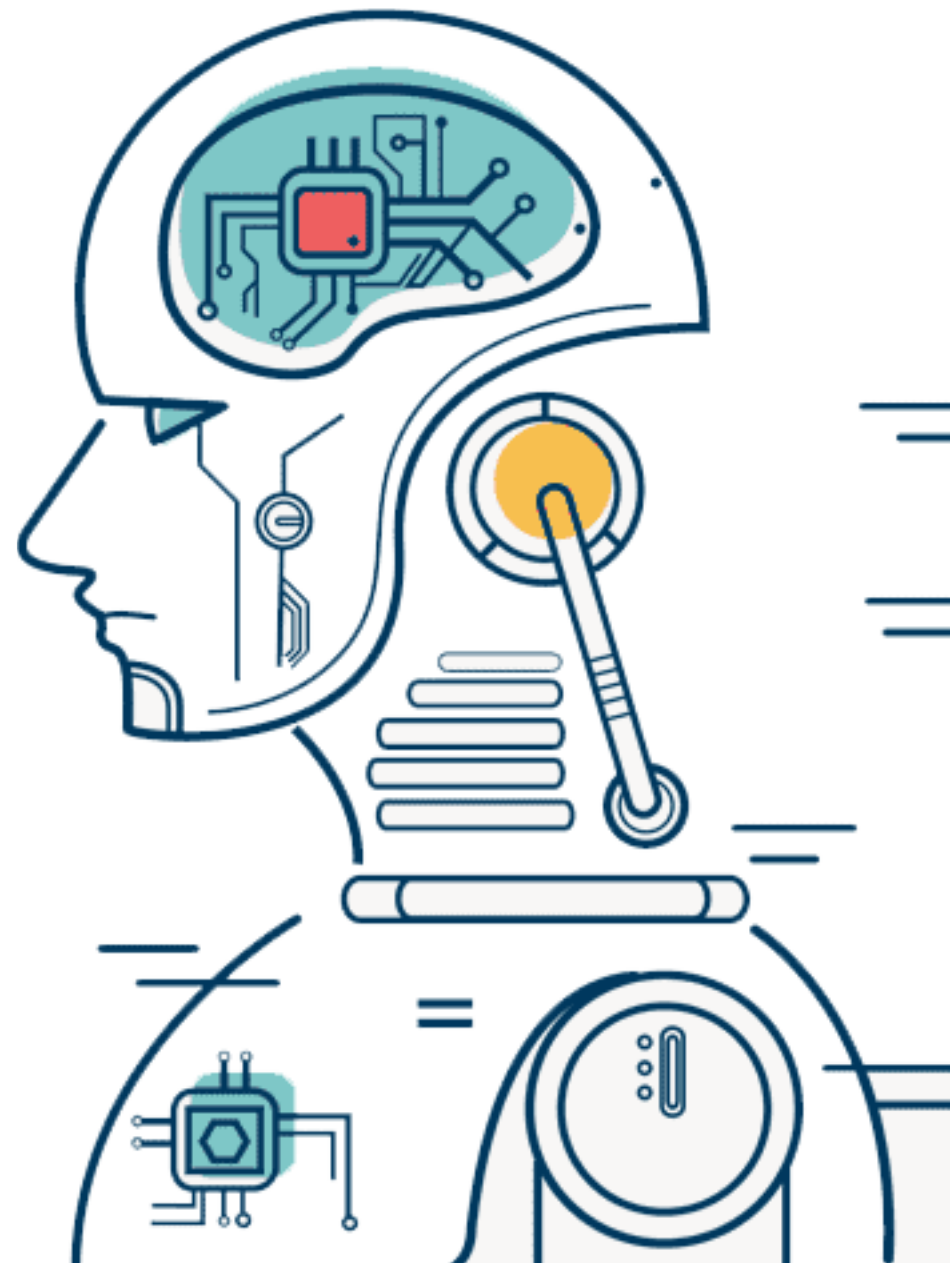
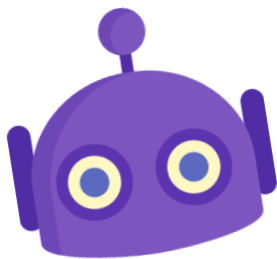


Deep Learning

Chapter 14 Autoencoder

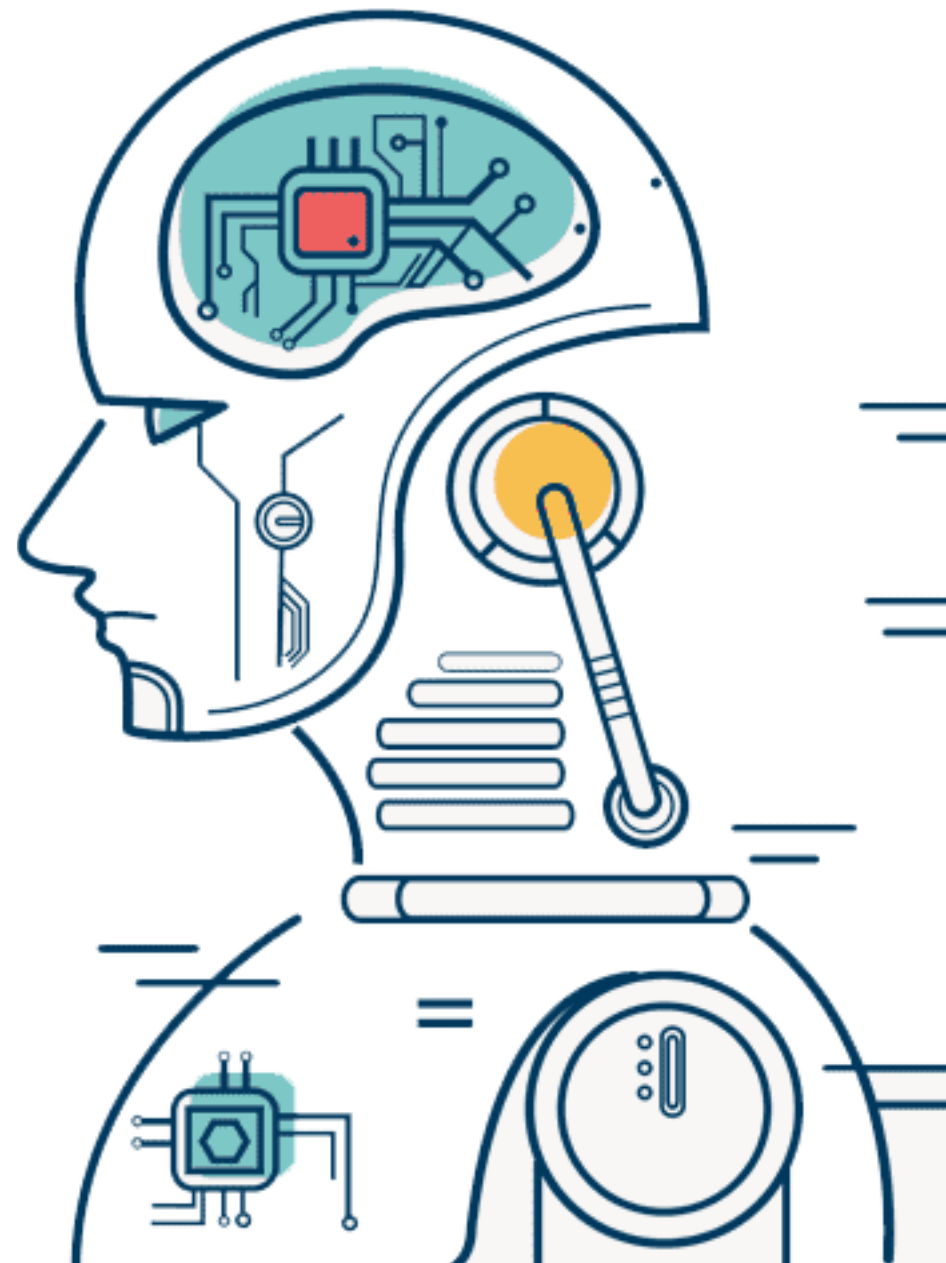


- **Autoencoder 개념을 이해 할 수 있다.**
- **Autoencoder를 활용할 수 있다.**

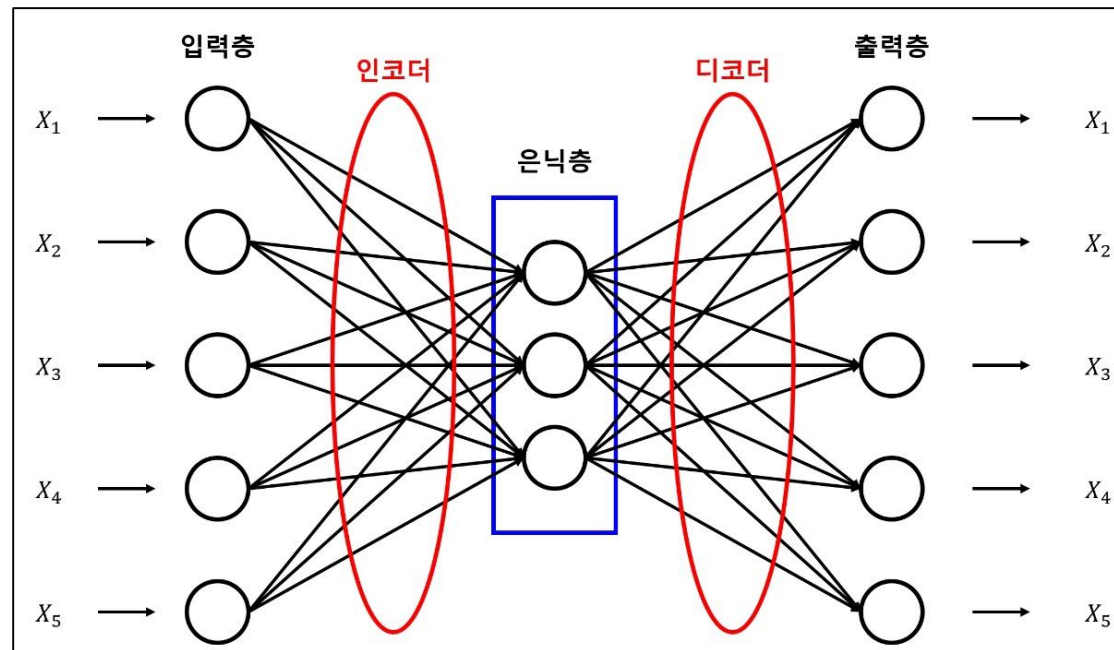


Autoencoder (AE)

Uncomplete autoencoder

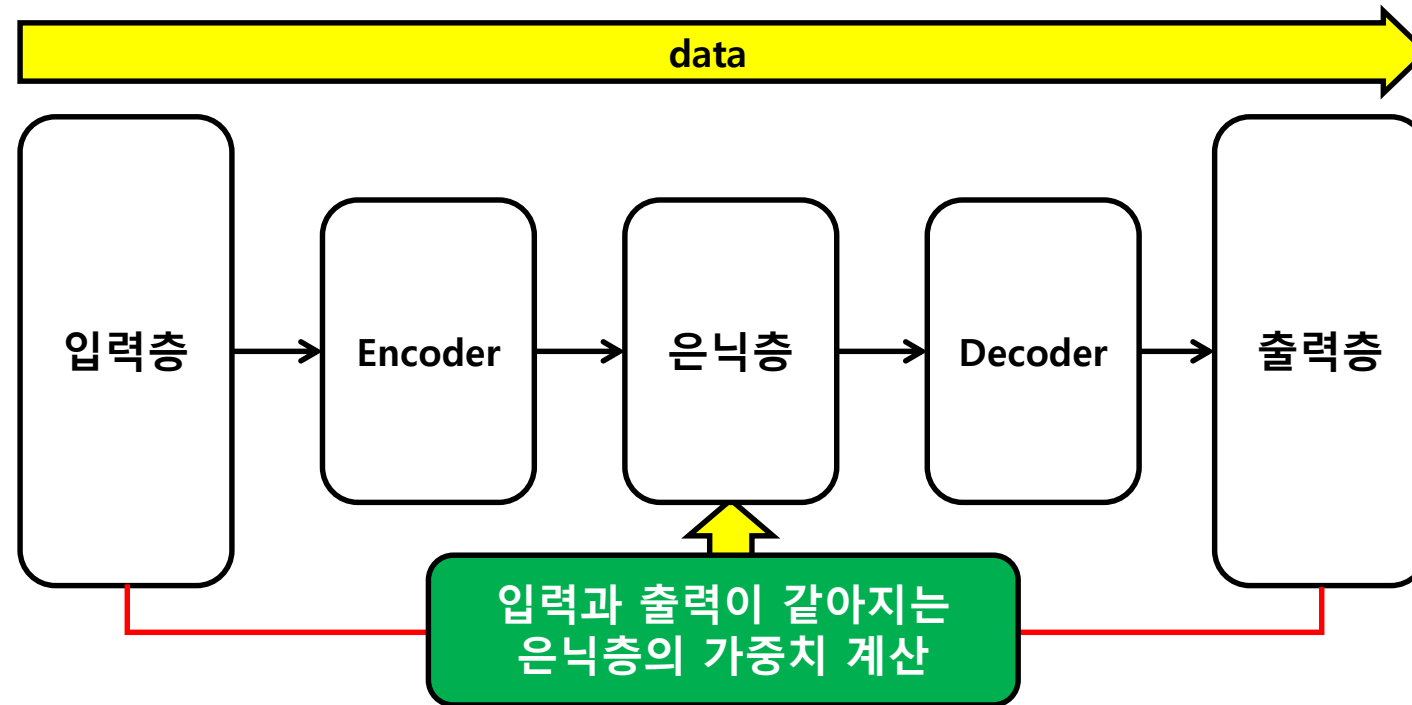


- **입력을 출력으로 복사하는 신경망 (Encoder와 Decoder로 구성) → 비지도 학습**
- 네트워크에 여러가지 방법으로 제약을 주어 복잡한 신경망을 생성
 - (예1) 은닉층의 뉴런 수를 입력층보다 작게하여 차원 축소 효과 (압축)
 - (예2) 잡음이 추가된 입력데이터에서 원본 입력을 복원할 수 있도록 네트워크를 학습



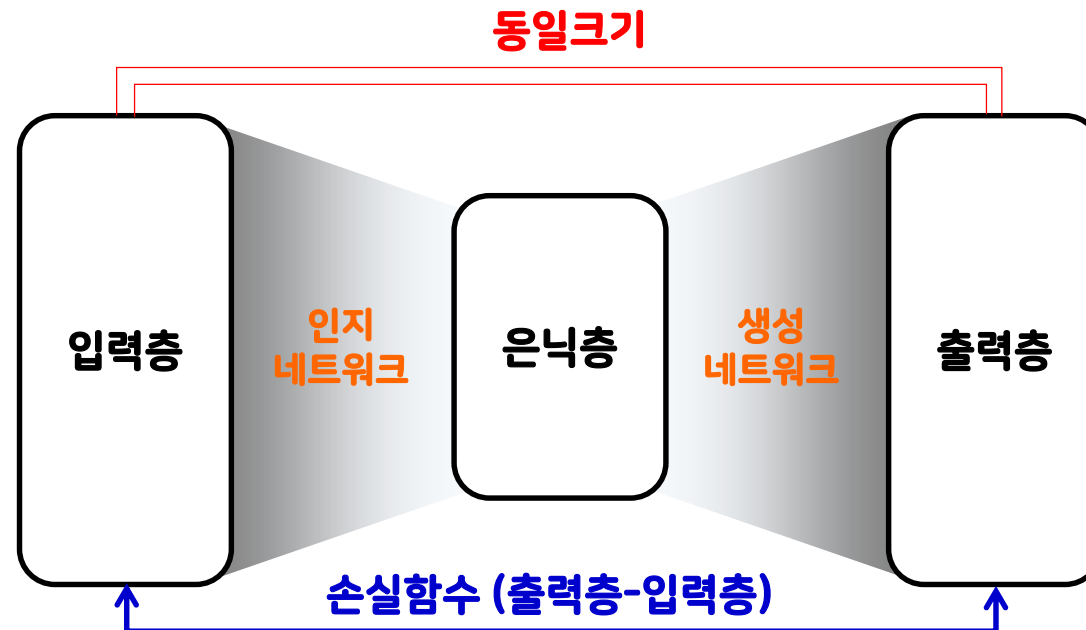
- Uncomplete autoencoder
- Sparse autoencoder
- Stacked autoencoder (Deep fully-connected autoencoder)
- Deep convolutional autoencoder
- Denoising autoencoder (잡음제거)
- Sequence-to-sequence autoencoder
- Variational autoencoder (변형, VAE)

- **Uncomplete AutoEncoder** : 은닉층의 노드가 입력층의 노드보다 작은 오토인코더
- 입력과 출력층이 동일해지도록 학습 → 입력 데이터에서 중요한 특성을 학습



Uncomplete AutoEncoder

- **Encoder** : 입력을 내부 표현으로 변환 → **인지 네트워크 (Recognition Network)**
- **Decoder** : 내부 표현을 출력으로 변환 → **생성 네트워크 (Generative Network)**
- MLP와의 차이점 : 입력층 수와 출력층 수가 동일 → 재구성 (Reconstruction)
- 손실함수의 계산 : **입력층과 출력층의 차이로 계산**



$$X \rightarrow X'$$

자기학습

신경망 설계

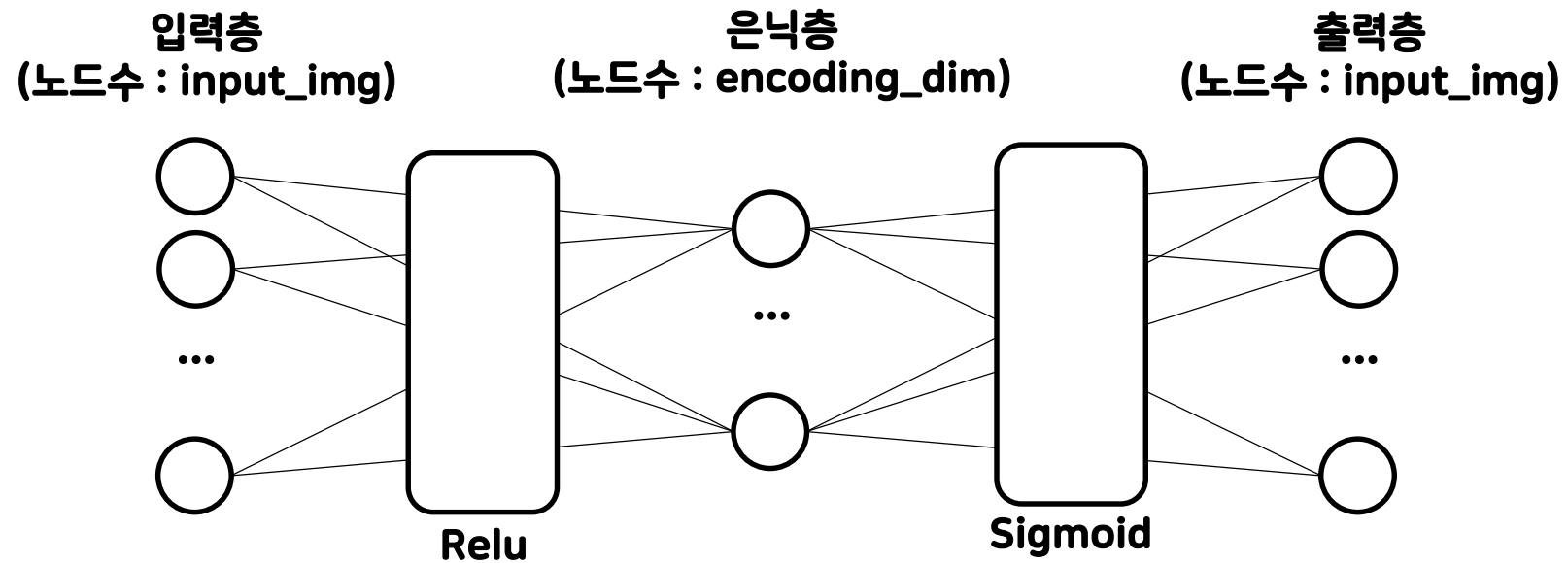
- 입력 데이터를 차원축소하여 은닉층에 전달

```
autoencoder_A = Sequential()

# 인코더 부분
# encoding_dim : 은닉층 노드수 - 인코더 표현(representation)의 크기
# input_img : 입력층의 노드수
autoencoder_A.add(Dense(encoding_dim, input_dim=input_img, activation='relu'))

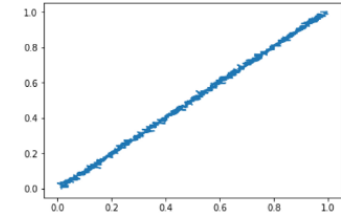
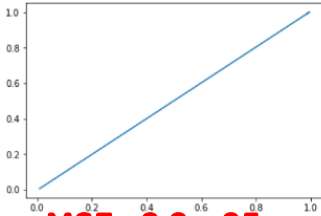
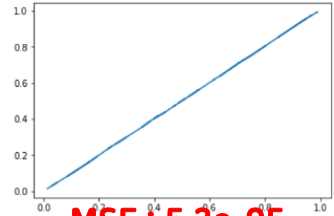
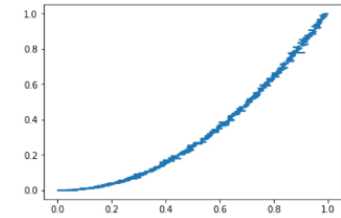
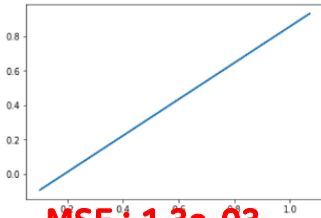
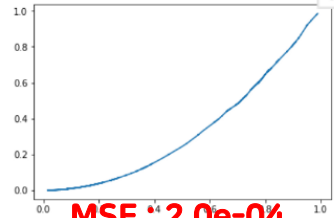
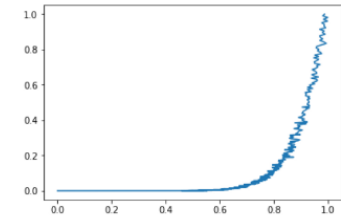
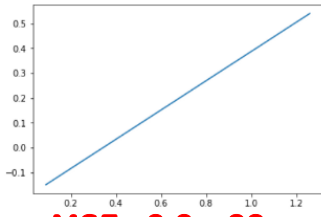
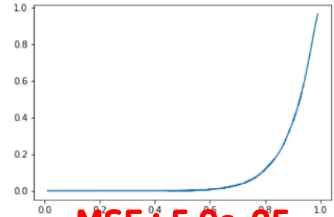
# 디코더 부분
autoencoder_A.add(Dense(input_img, activation='sigmoid'))
```

신경망 설계

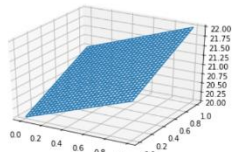
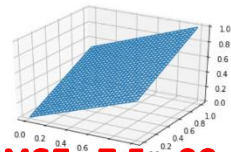
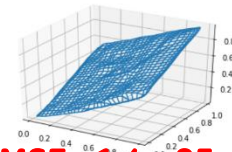
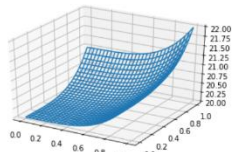
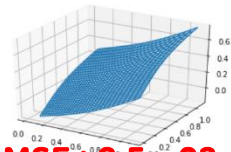
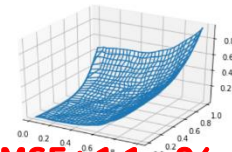


특징공간 vs PCA 재구성 vs AE 재구성

- 일반적인 차원 축소 방법 : PCA, AutoEncoder
- PCA (주성분 분석) : 직교변환투영을 통해 분산이 가장 큰 벡터를 계산
→ 선형 함수 모델링, 고속, 상관관계 없음
- AE : 비선형 함수 모델링, 상관관계 있음, 과적합 (많은 매개변수)

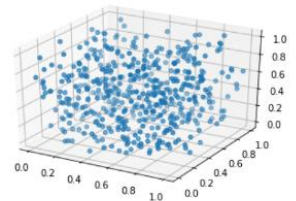
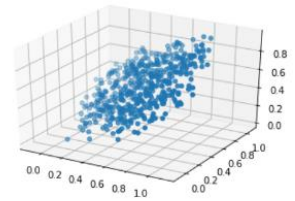
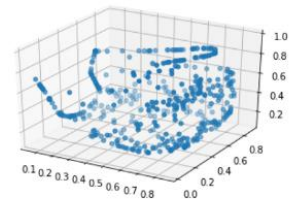
Function	Feature Space	PCA Reconstruction	Auto Encoder Reconstruction
$y=mx+c$		 MSE : 2.9e-05	 MSE : 5.3e-05
$y=mx^2+c$		 MSE : 1.3e-03	 MSE : 2.0e-04
$y=mx^8+c$		 MSE : 8.8e-03	 MSE : 5.0e-05

특징공간 vs PCA 재구성 vs AE 재구성

Function	Feature Space	PCA Reconstruction	Auto Encoder Reconstruction
Plane		 MSE : 7.5e-32	 MSE : 1.4e-05
Curved Surface		 MSE : 2.5e-03	 MSE : 1.1e-04

3차원 데이터

- 평면인 경우는 PCA는 100% 재구성되나 곡면인 경우는 정보 손실
- AE는 평면, 곡면 모두 잘 동작

	Feature Space	PCA Reconstruction	Auto Encoder Reconstruction
Random Data			
Reconstruction Cost (MSE) ▼		0.024	0.010

상관관계가 없는 무작위 데이터

- PCA는 2차원 구조가 없기 때문에 정보손실
- AE는 잘 동작하지 않음

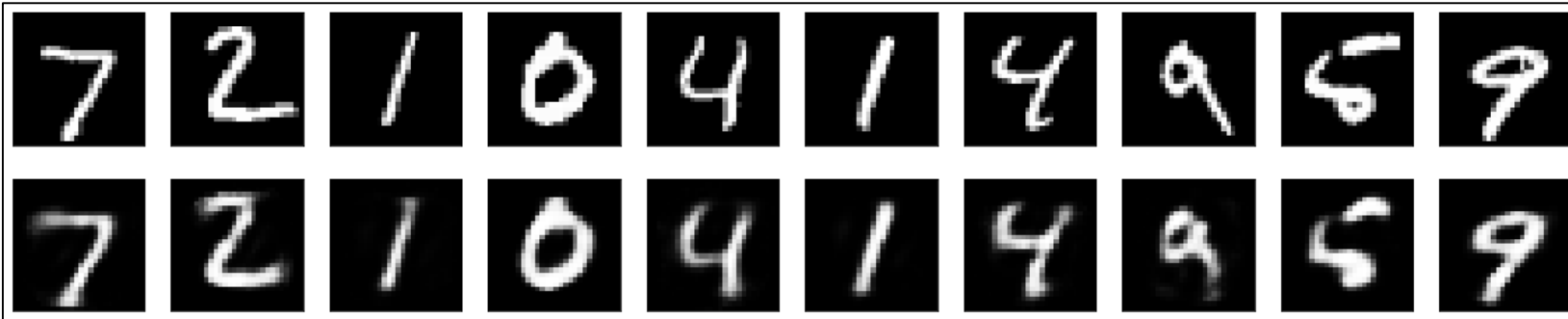
- 오토 인코더 컴파일 및 학습하기

```
autoencoder_A.compile(optimizer="adam", loss="binary_crossentropy")  
  
autoencoder_A.fit(x_train, x_train, epochs=20, batch_size=256, shuffle=True,  
validation_data=(x_test, x_test))
```

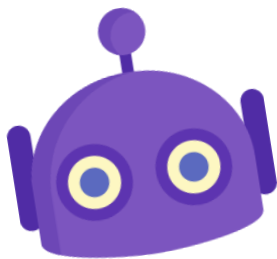
실행 결과

- MNIST 적용 결과 : **loss: 0.0936 val_loss: 0.0923**

원본 데이터

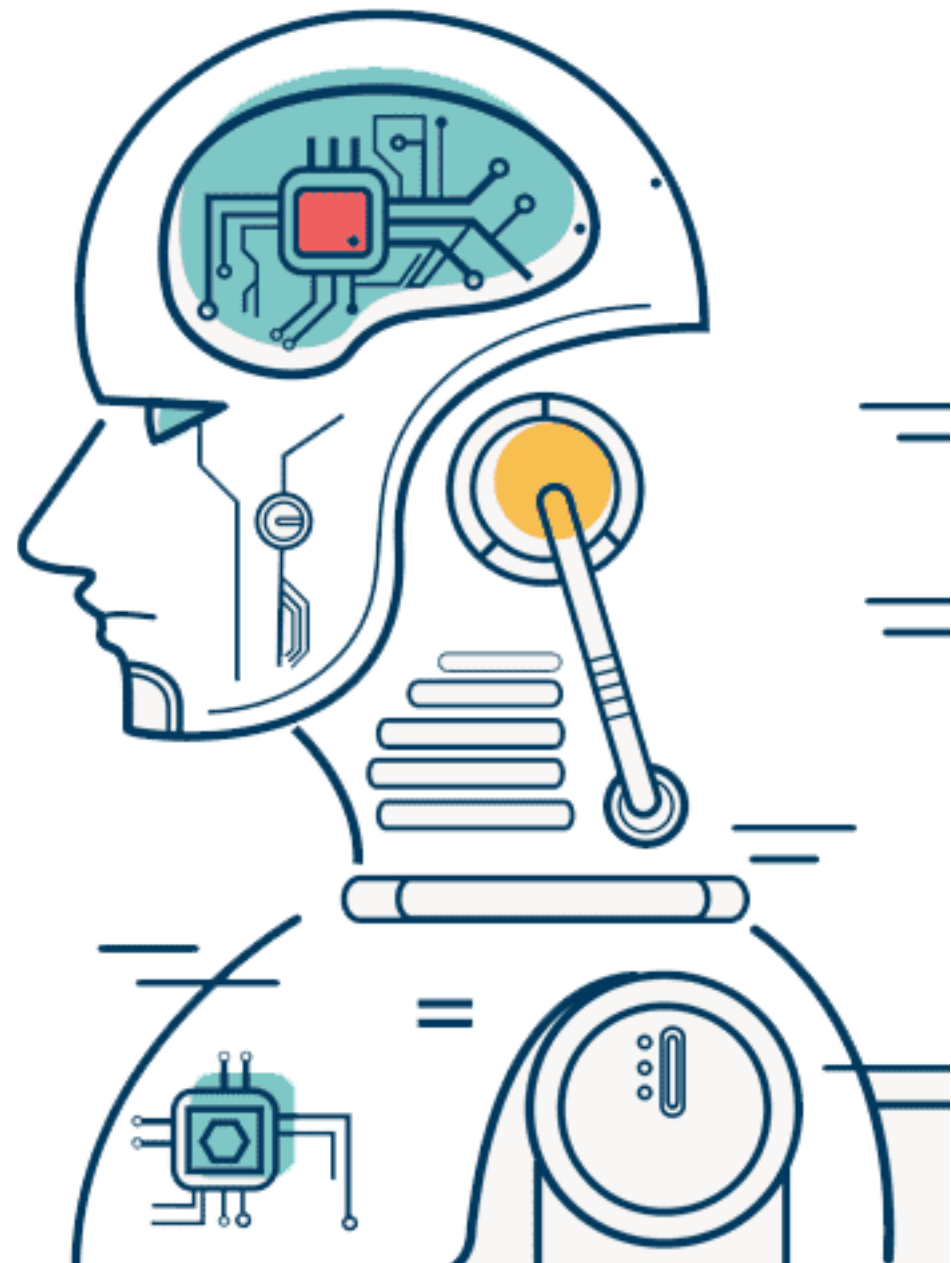


디코딩된 데이터

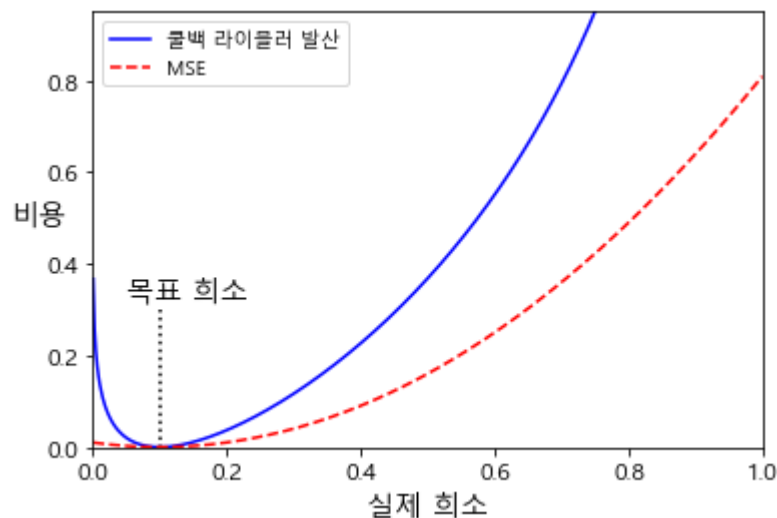


Autoencoder (AE)

Sparse autoencoder



- 뉴런의 수 결정 : 은닉층의 각 뉴런의 평균 활성화 정도를 계산하고 손실함수에 희소 손실을 추가하여 뉴런이 활성화되는 것을 규제 → MSE, L1 규제, KL 발산 등
- MSE 대신에 L1 규제나 KL 발산을 사용하여 희소성(Sparsity)을 적용한 근사화 → 은닉층 (코딩층)에 활성화되는 뉴런의 수를 감소시킴



$$Loss = MSE + sparsity_weight \times sparsity_loss$$

- **Sparse data (희소 데이터)** : 전체 공간에 비해 데이터가 있는 공간이 매우 협소한 데이터 ↔ **Dense data (밀집 데이터)**
- **Sparse data의 예** : 방문의 여달음을 체크하는 센서가 있다고 할 때 센서 값은 간헐적으로 획득, 특정 주제에 대한 정보
- **Dense data의 예** : 풍속을 기록하는 센서가 있다고 할 때 센서 값은 지속적으로 획득, 주제에 관계없는 여러가지 정보

신경망 설계

```
# 인코더 부분
# L1 규제를 사용하여 sparsity 적용
autoencoder_B.add(Dense(encoding_dim, input_dim=input_img, activation='relu',
activity_regularizer=regularizers.l1(10e-7)))

# 디코더 부분
autoencoder_B.add(Dense(input_img, activation='sigmoid'))
```

신경망 설계

```
autoencoder_B.compile(optimizer="adam", loss="binary_crossentropy")  
  
autoencoder_B.fit(x_train, x_train, epochs=50, batch_size=256, shuffle=True,  
validation_data=(x_test, x_test))
```

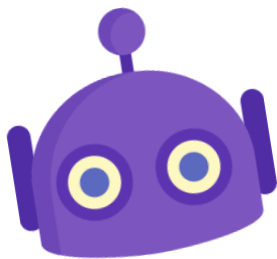
실행 결과

- MNIST 적용 결과 : **loss: 0.0929** **val_loss: 0.0917**

원본 데이터

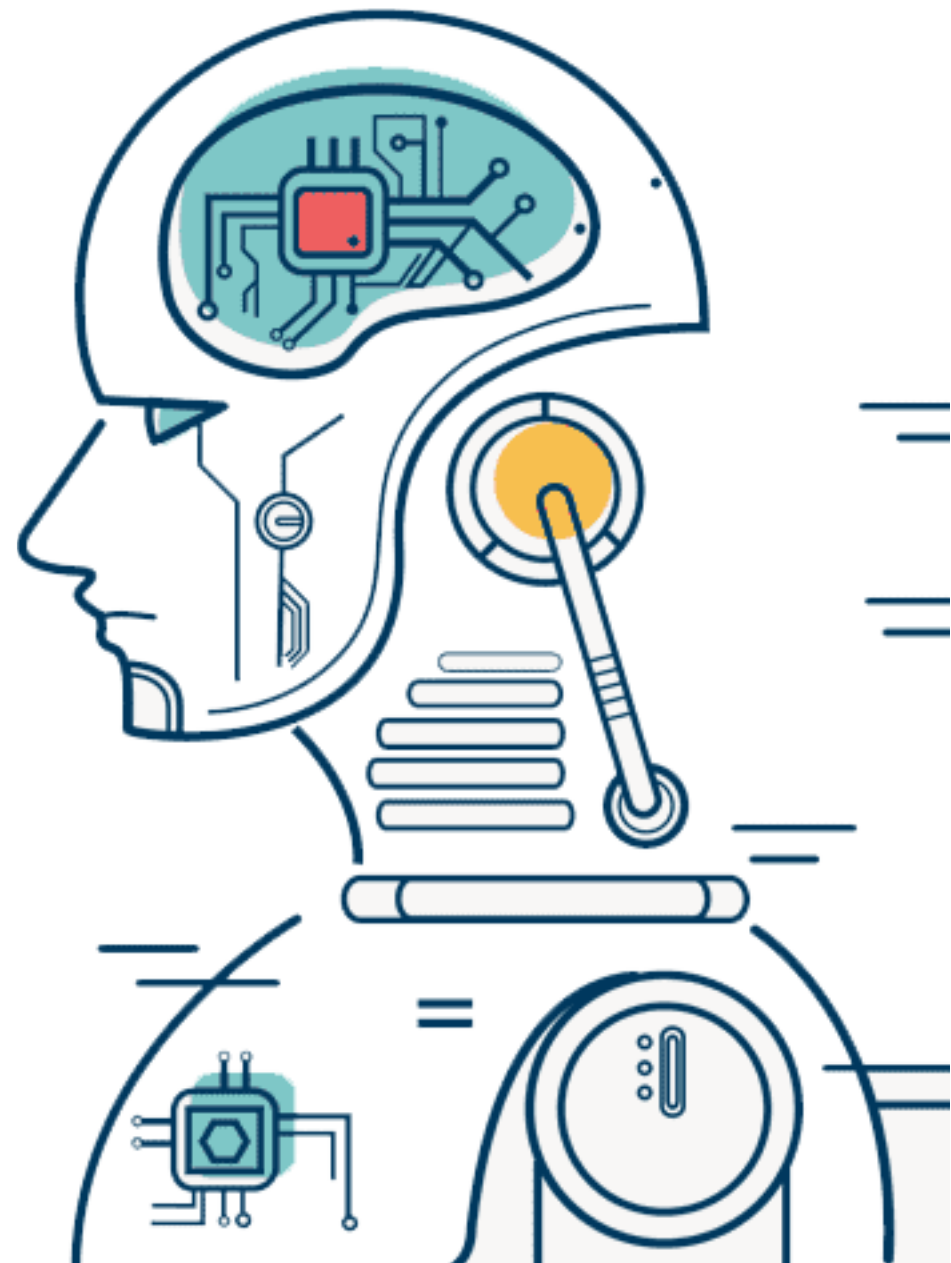


디코딩된 데이터



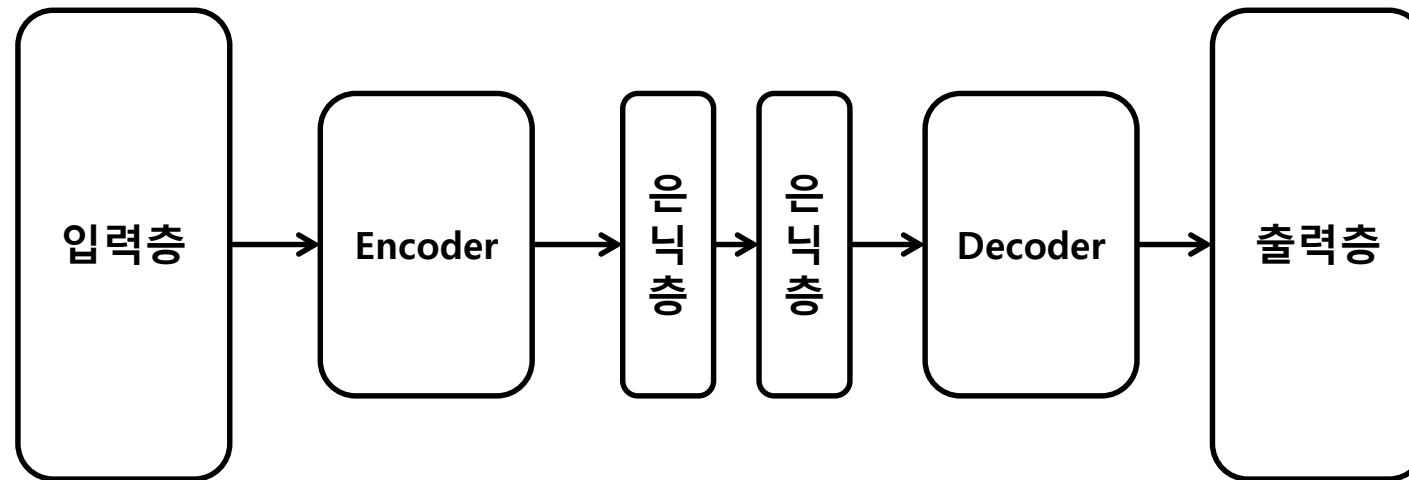
Autoencoder (AE)

Stacked AutoEncoder



Stacked AutoEncoder

- AutoEncoder (AE)가 은닉층이 1개인데 반해 Stacked AutoEncoder는 은닉층이 여러 개인 AutoEncoder



신경망 설계

- 은닉층을 추가한 것 → 인코더와 디코더의 노드 수는 대칭이 되어야 함

```
autoencoder_C = Sequential()
```

인코더 부분

```
autoencoder_C.add(Dense(128, input_dim=input_img, activation='relu'))
```

```
autoencoder_C.add(Dense(64, activation='relu'))
```

```
autoencoder_C.add(Dense(32, activation='relu'))
```

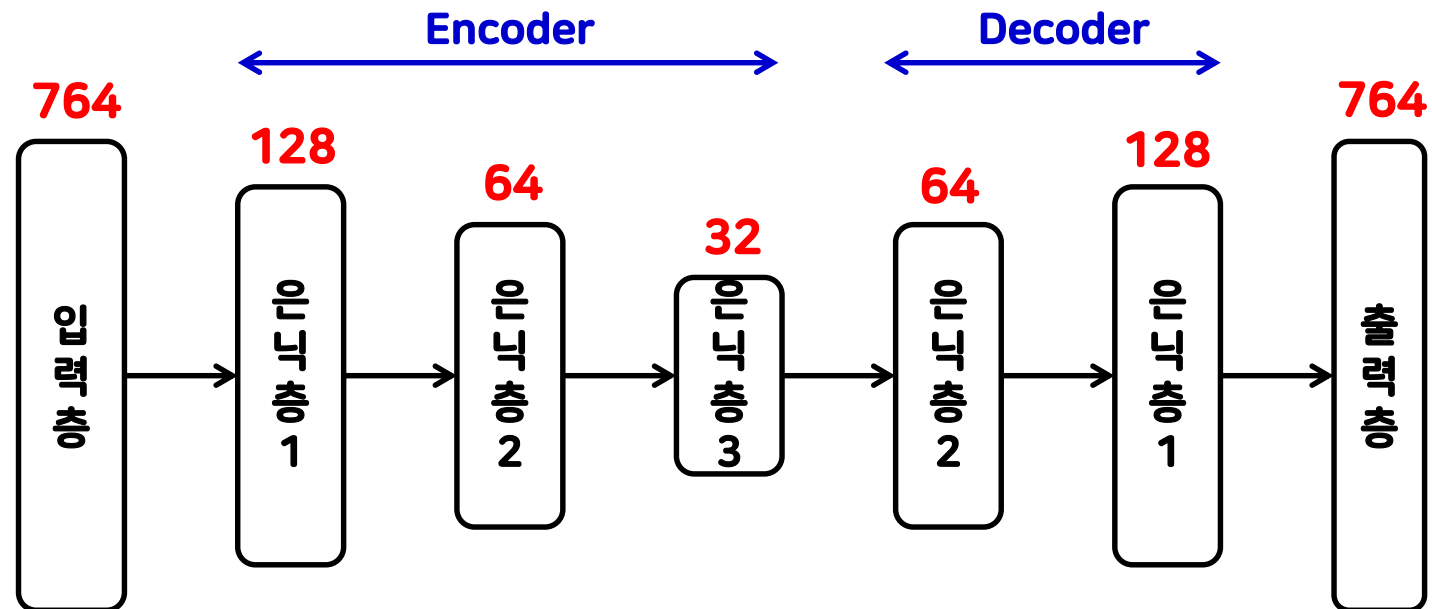
디코더 부분

```
autoencoder_C.add(Dense(64, activation='relu'))
```

```
autoencoder_C.add(Dense(128, activation='relu'))
```

```
autoencoder_C.add(Dense(input_img, activation='sigmoid'))
```

신경망 설계



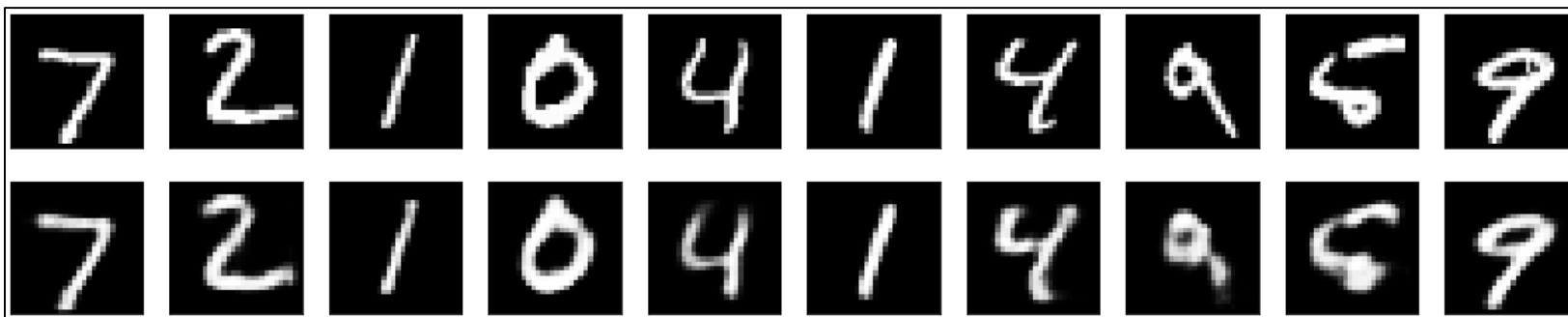
신경망 설계

```
autoencoder_C.compile(optimizer='adam', loss='binary_crossentropy')  
  
autoencoder_C.fit(x_train, x_train, epochs=20, batch_size=256, shuffle=True,  
validation_data=(x_test, x_test))
```

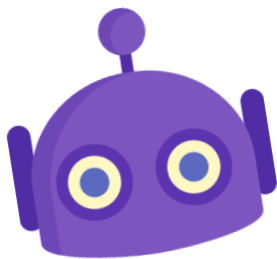
실행 결과

- MNIST 적용 결과 : **loss: 0.0926 val_loss: 0.0914**

원본 데이터

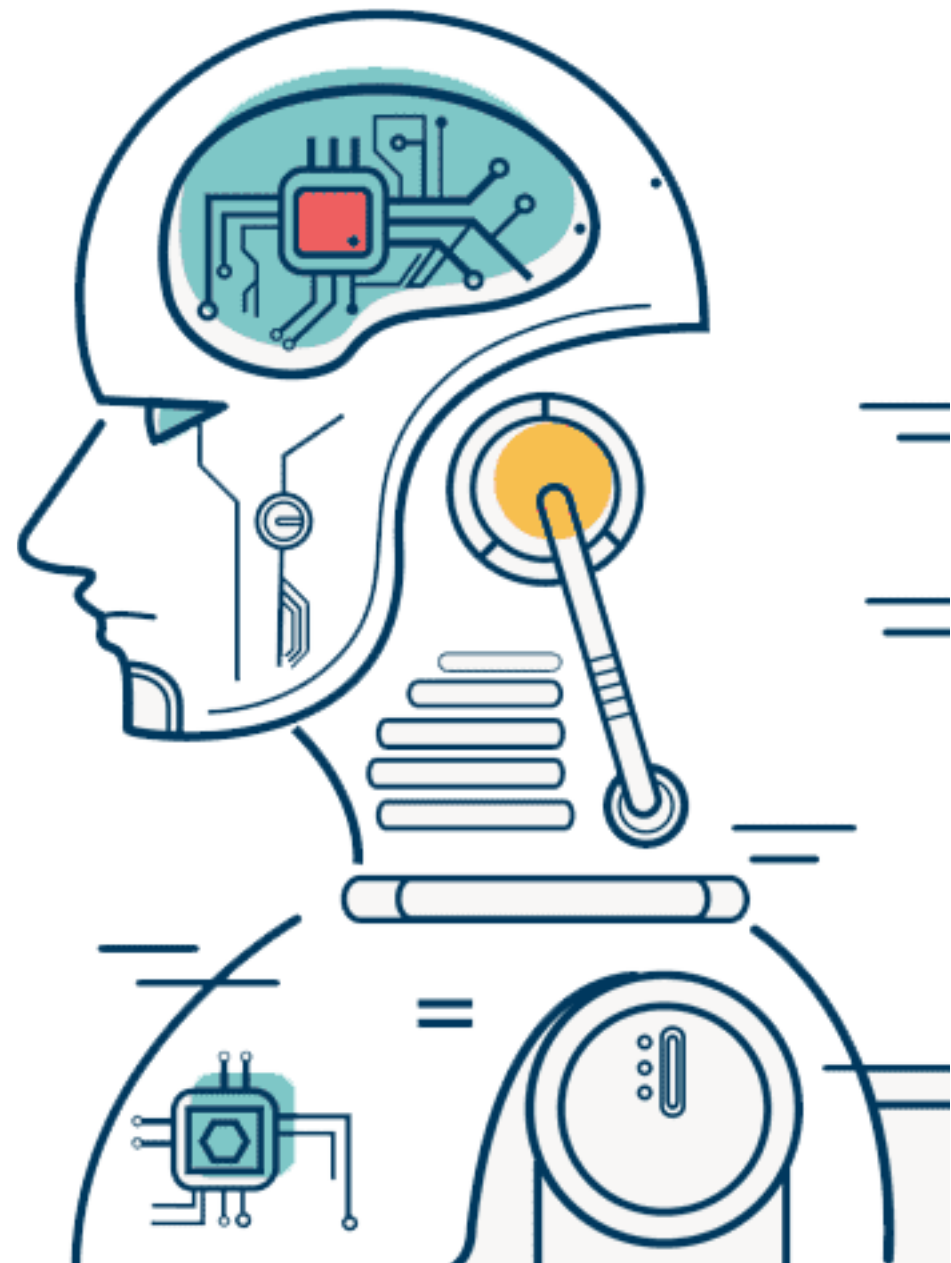


디코딩된 데이터



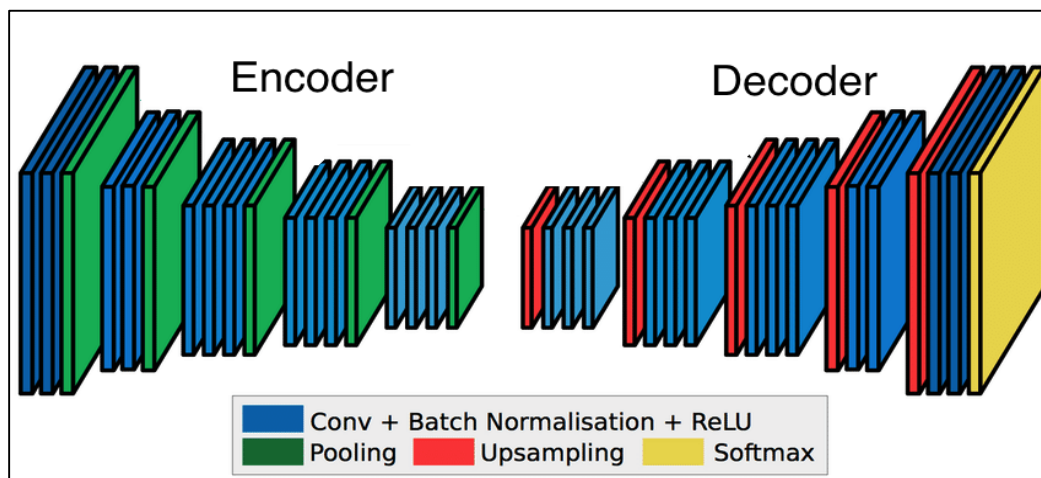
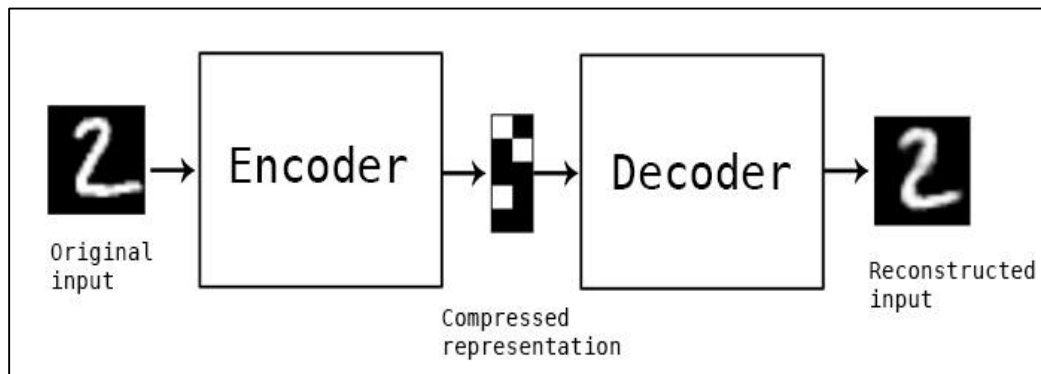
Autoencoder (AE)

Deep convolutional autoencoder



신경망 설계

- 은닉층에 CNN을 적용한 모델 → 입력데이터가 이미지인 경우 주로 사용



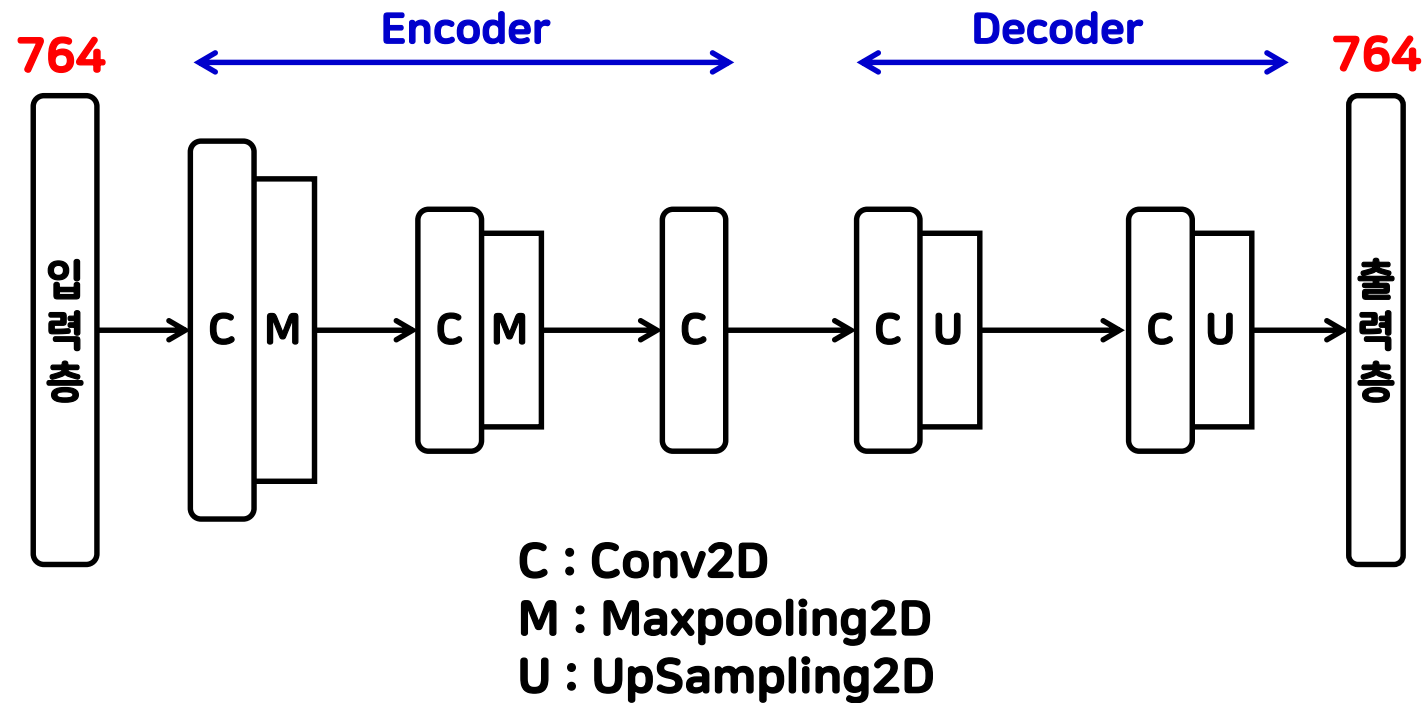
신경망 설계

```
autoencoder_D = Sequential()

# 인코더 부분 - 차원 축소
# 흑백이미지 1, 칼라이미지 3
autoencoder_D.add(Conv2D(16, (3, 3), padding='same', input_shape=(28, 28, 1), activation='relu'))
autoencoder_D.add(MaxPooling2D((2, 2), padding='same'))
autoencoder_D.add(Conv2D(8, (3, 3), padding='same', activation='relu'))
autoencoder_D.add(MaxPooling2D((2, 2), padding='same'))
autoencoder_D.add(Conv2D(8, (3, 3), padding='same', activation='relu'))

# 디코더 부분 - 차원 증가
# 이 시점에서 표현(representatoin)은 (7,7,8) 즉, 584 차원
autoencoder_D.add(Conv2D(8, (3, 3), padding='same', activation='relu'))
# MaxPooling2D을 반대로 처리
autoencoder_D.add(UpSampling2D((2, 2)))
autoencoder_D.add(Conv2D(8, (3, 3), padding='same', activation='relu'))
autoencoder_D.add(UpSampling2D((2, 2)))
autoencoder_D.add(Conv2D(1, (3, 3), padding='same', activation='sigmoid'))
```

신경망 설계



신경망 설계

`autoencoder_D.summary()`

Model: "sequential_19"

Layer (type)	Output Shape	Param #
conv2d_55 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_23 (MaxPooling)	(None, 14, 14, 16)	0
conv2d_56 (Conv2D)	(None, 14, 14, 8)	1160
max_pooling2d_24 (MaxPooling)	(None, 7, 7, 8)	0
conv2d_57 (Conv2D)	(None, 7, 7, 8)	584
conv2d_58 (Conv2D)	(None, 7, 7, 8)	584
up_sampling2d_24 (UpSampling)	(None, 14, 14, 8)	0
conv2d_59 (Conv2D)	(None, 14, 14, 8)	584
up_sampling2d_25 (UpSampling)	(None, 28, 28, 8)	0
conv2d_60 (Conv2D)	(None, 28, 28, 1)	73

Total params: 3,145

Trainable params: 3,145

Non-trainable params: 0

신경망 설계

- 입력 이미지 크기 변환

```
from keras.datasets import mnist
import numpy as np

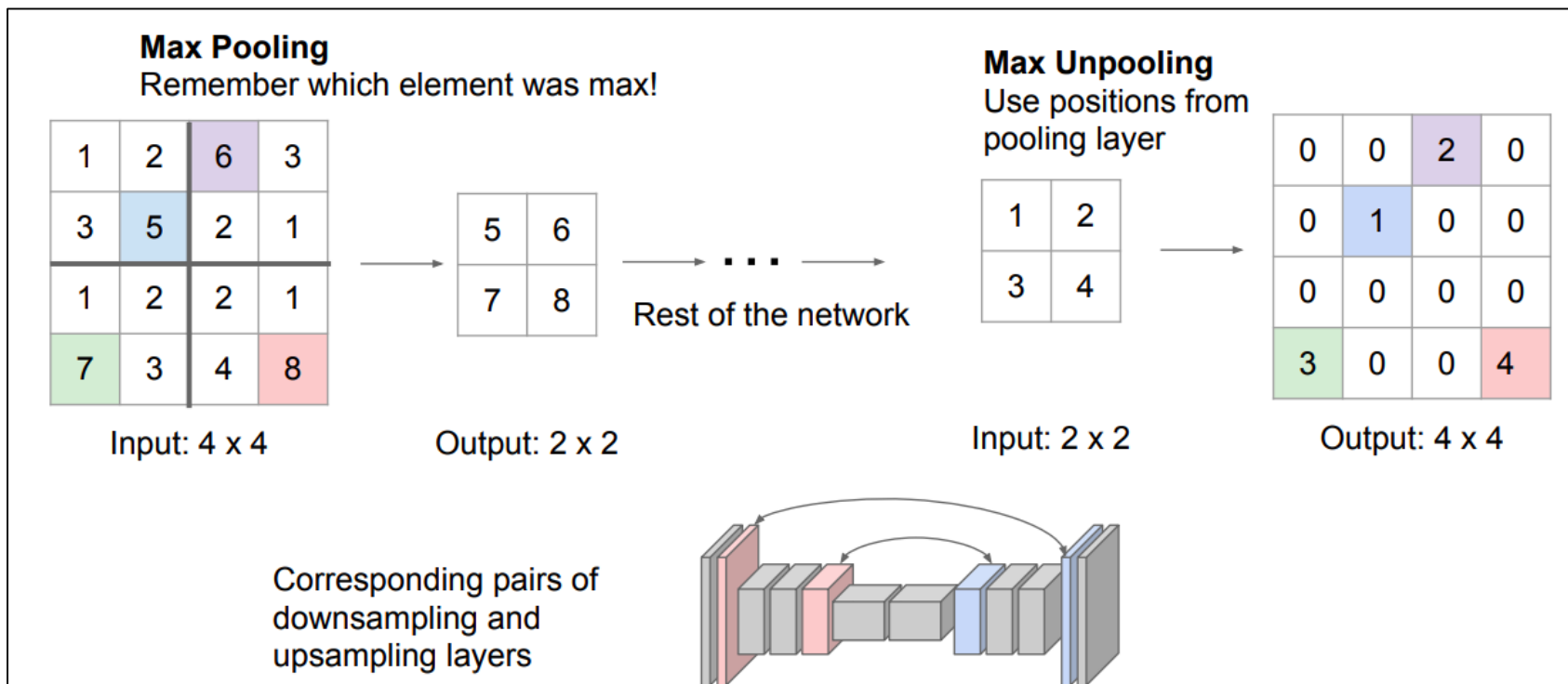
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

```
x_train.shape, x_test.shape
```

```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

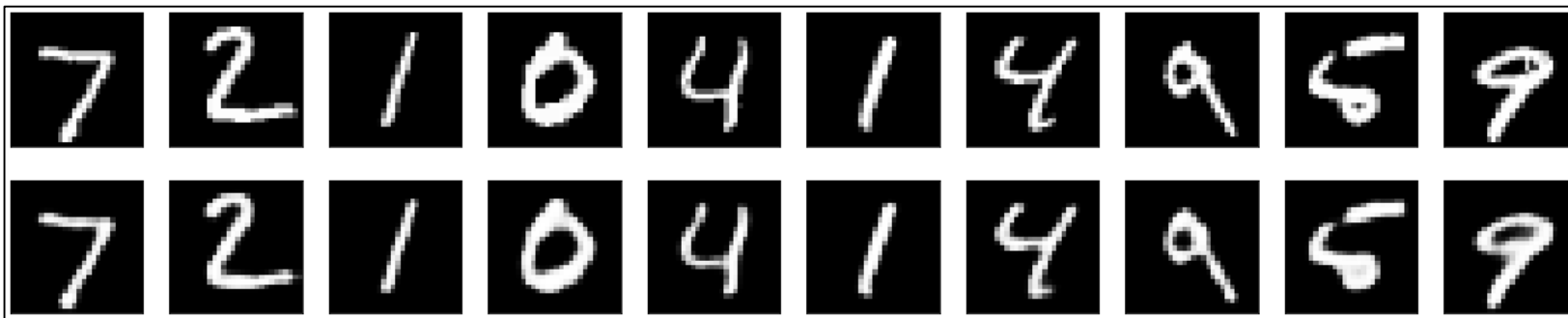

신경망 설계



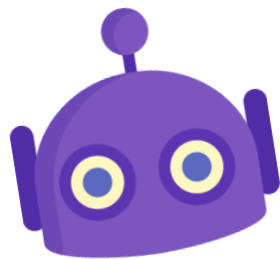
실행 결과

- MNIST 적용 결과 : **loss: 0.0729 val_loss: 0.0722**

원본 데이터

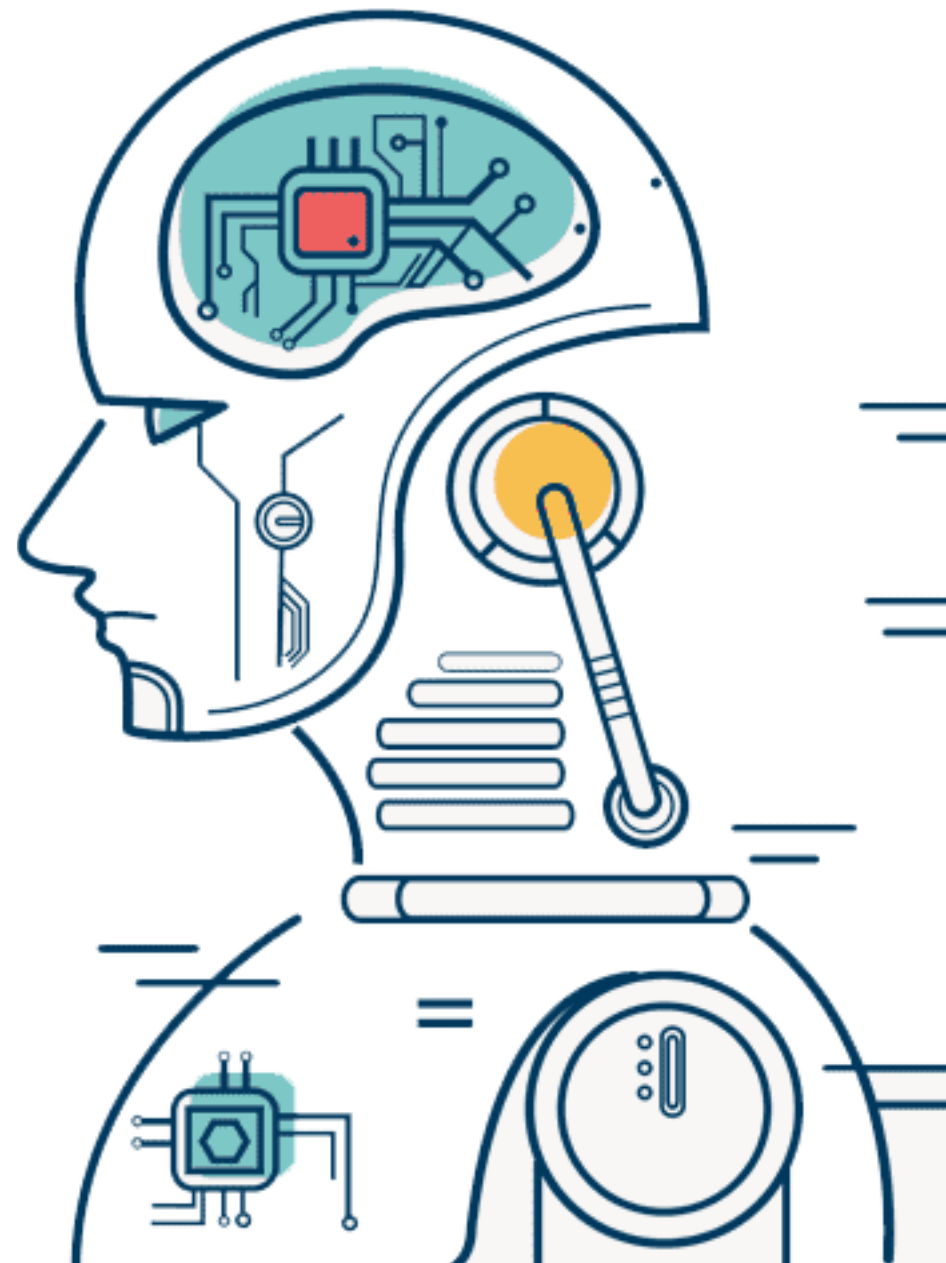


디코딩된 데이터



Autoencoder (AE)

Denoising Autoencoder



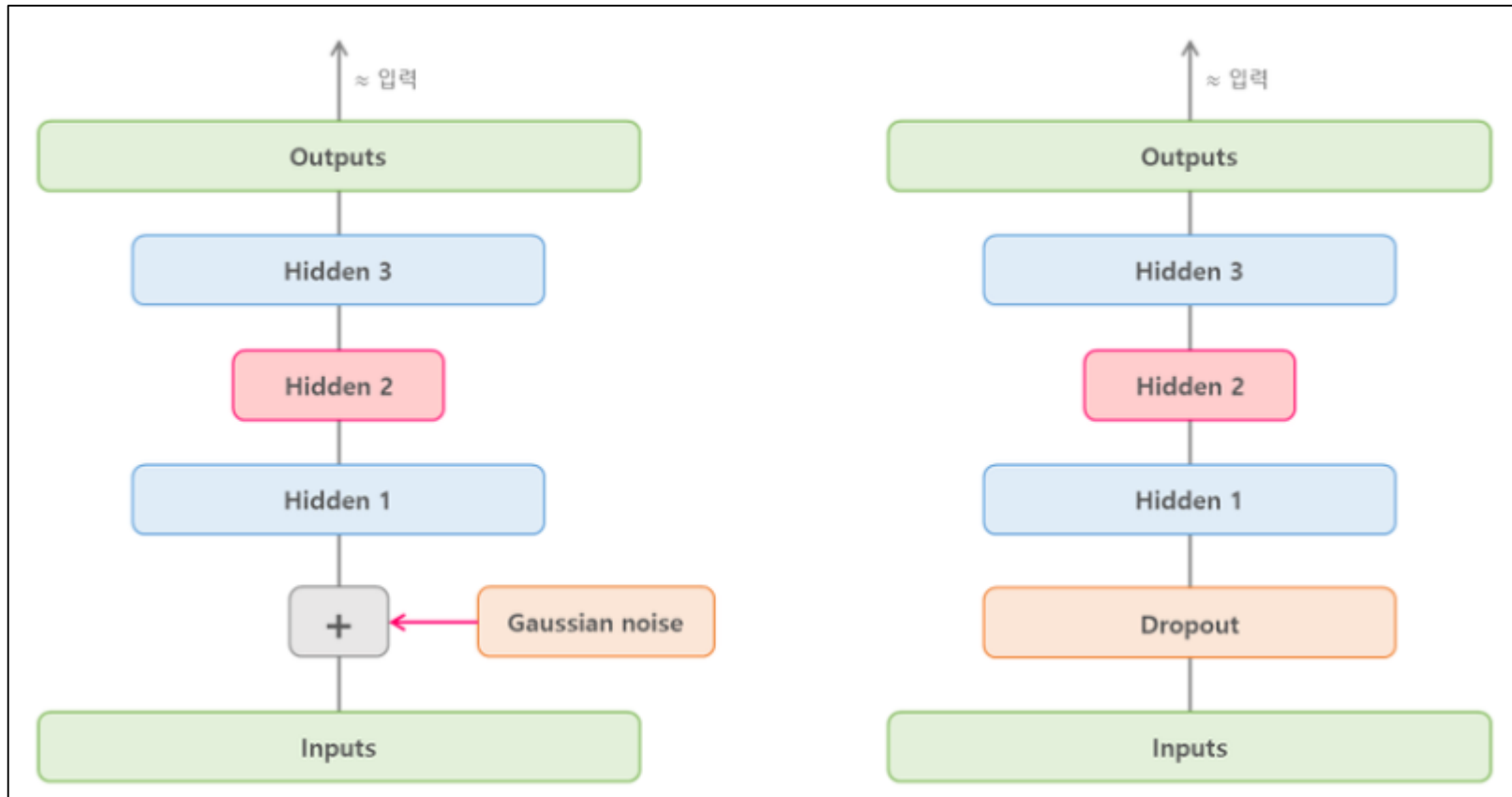
- 잡음이 포함된 이미지에서 잡음을 제거하여 원본 이미지 복원이 목적

원본 데이터



잡음이 추가된 데이터

- 잡음 추가 : Gaussian 잡음을 추가하거나 Dropout으로 랜덤하게 입력노드를 꺼서 발생



신경망 설계

- 잡음 추가

잡음비율

```
noise_factor = 0.5
```

정규분포로부터 무작위 샘플을 추출

normal(이산평균, 이산표준편차, 데이터수)

```
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,  
                                                         size=x_train.shape)
```

```
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0,  
                                                         size=x_test.shape)
```

clip(배열, 최소값 기준, 최대값 기준) : 최소/최대 범위를 벗어나는 값을 최소/최대값으로 치환

```
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
```

```
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

신경망 설계

- Convolutional Autoencoder에 비해 재구성된 이미지의 질을 향상시키려면 Layer당 더 많은 필터가 필요

인코더 부분

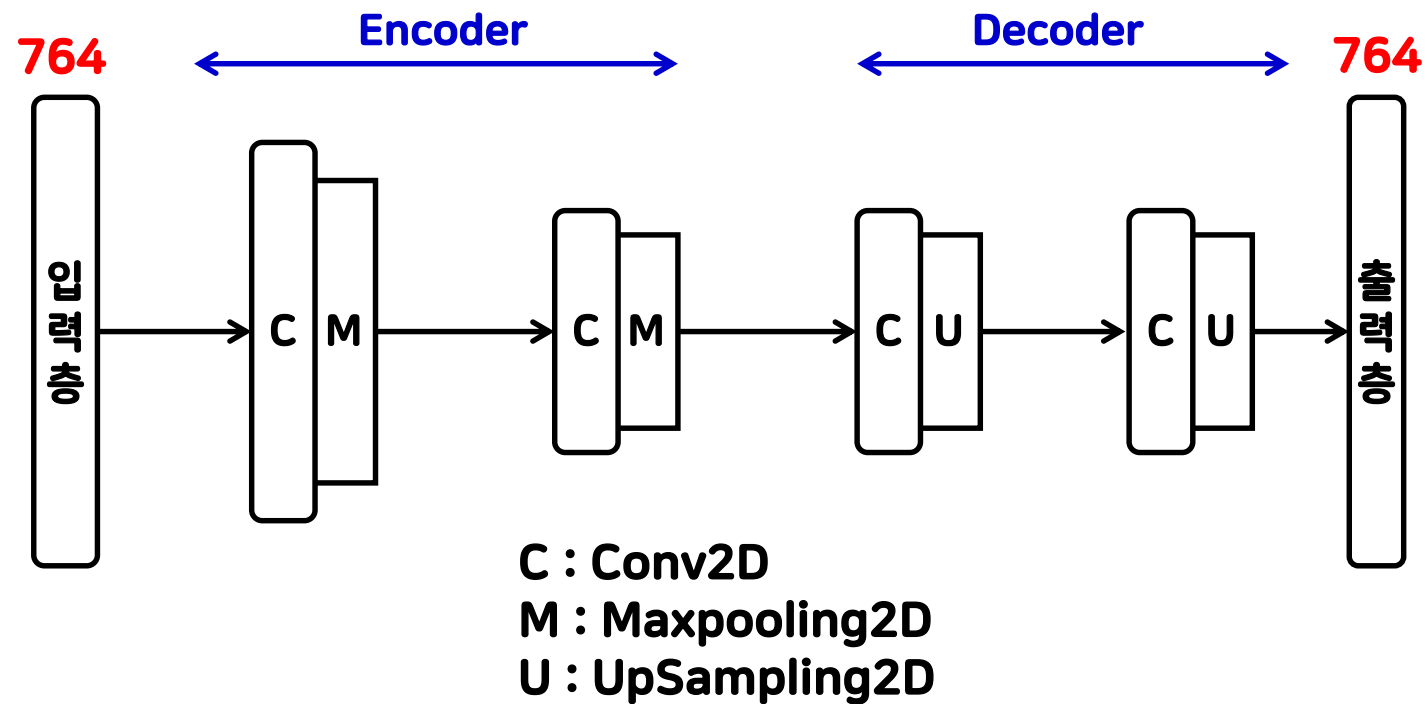
```
autoencoder_E.add(Conv2D(32, (3, 3), padding='same', input_shape=(28, 28, 1),  
                        activation='relu'))  
autoencoder_E.add(MaxPooling2D((2, 2), padding='same'))  
autoencoder_E.add(Conv2D(32, (3, 3), padding='same', activation='relu'))  
autoencoder_E.add(MaxPooling2D((2, 2), padding='same'))
```

디코더 부분

이 시점에서 표현(representation)은 (7,7,32)

```
autoencoder_E.add(Conv2D(32, (3, 3), padding='same', activation='relu'))  
autoencoder_E.add(UpSampling2D((2, 2)))  
autoencoder_E.add(Conv2D(32, (3, 3), padding='same', activation='relu'))  
autoencoder_E.add(UpSampling2D((2, 2)))  
autoencoder_E.add(Conv2D(1, (3, 3), padding='same', activation='sigmoid'))
```

신경망 설계



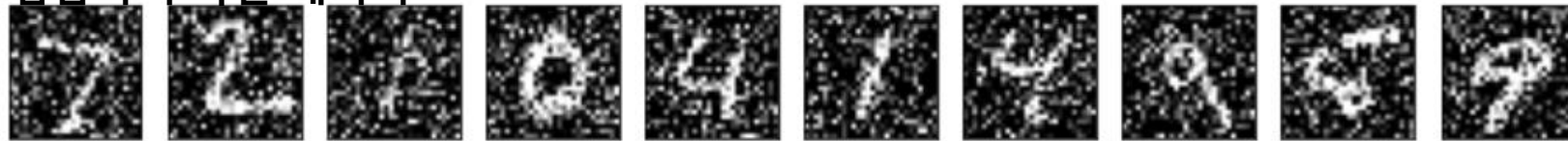
실행 결과

- MNIST 적용 결과 : **loss: 0.0956 val_loss: 0.0949**

원본 데이터



잡음이 추가된 데이터



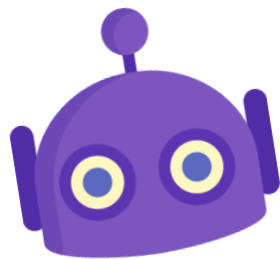
잡음이 제거된 데이터

- Dirty Documents의 Noise 제거 (데이터 : <https://www.kaggle.com/c/denoising-dirty-documents>)
- 소스코드 : <https://www.kaggle.com/ahmedpyarali/autoencoded-denoising>

There exist several methods to design forms with fields to fields may be surrounded by bounding boxes, by light rectangles or methods specify where to write and, therefore, minimize the effect with other parts of the form. These guides can be located on a sheet is located below the form or they can be printed directly on the form a separate sheet is much better from the point of view of the user but requires giving more instructions and, more importantly, rest this type of acquisition is used. Guiding rulers printed on the used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, be taken into account: The best way to print these light rectangles

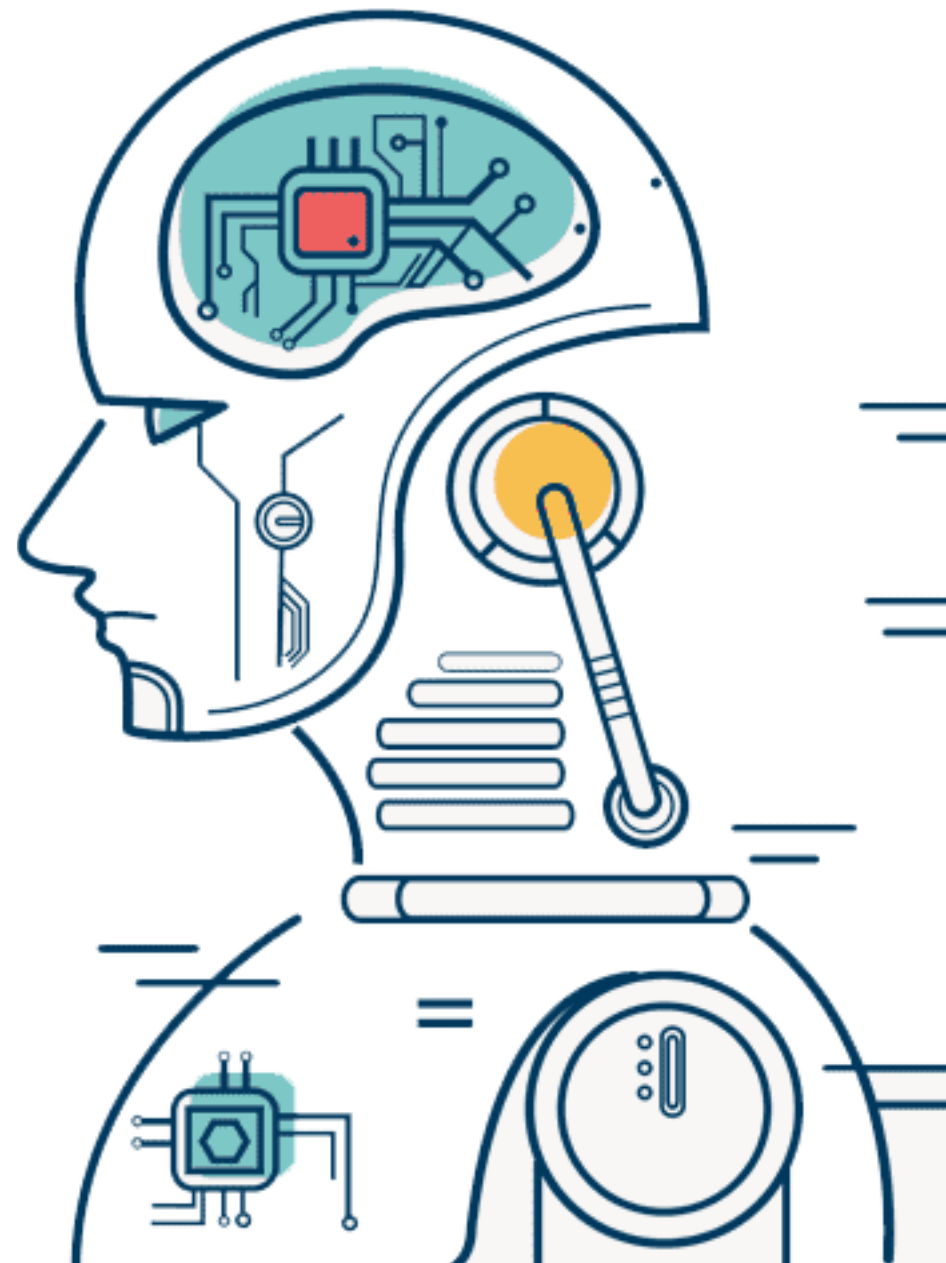


There exist several methods to design forms with fields to fields may be surrounded by bounding boxes, by light rectangles or methods specify where to write and, therefore, minimize the effect with other parts of the form. These guides can be located on a sheet is located below the form or they can be printed directly on the form a separate sheet is much better from the point of view of the user but requires giving more instructions and, more importantly, rest this type of acquisition is used. Guiding rulers printed on the used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, be taken into account: The best way to print these light rectangles



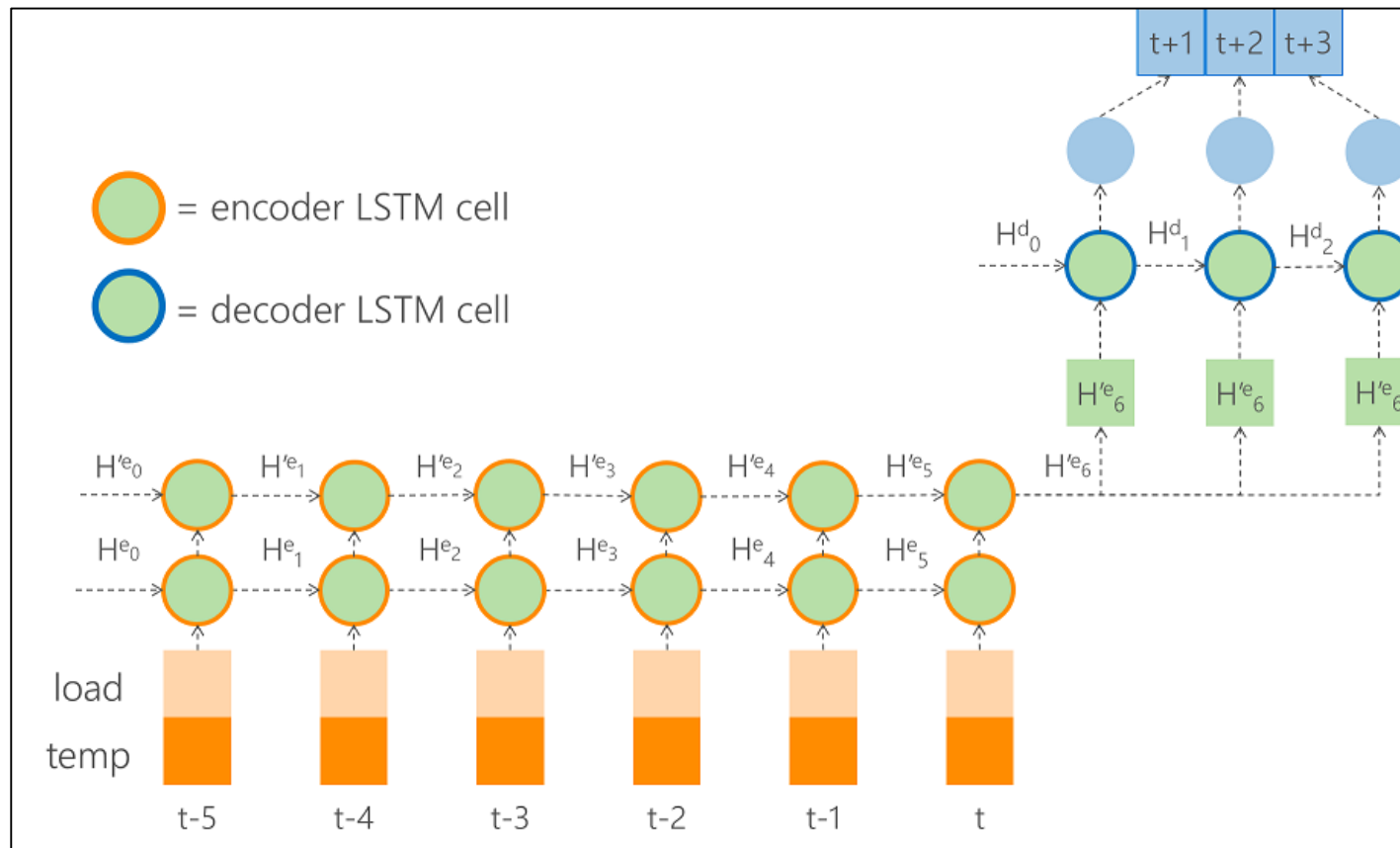
Autoencoder (AE)

Sequence-to-sequence Autoencoder



- 시계열 데이터에서 활용
- 벡터나 2D 이미지가 아닌 입력값이 연속적이라면, 인코더와 디코더를 시간 구조로 잡은 모델이 필요
- LSTM 기반의 autoencoder를 만들기 위해서는
 - 먼저 LSTM 인코더를 사용하여 입력 시퀀스를 전체 시퀀스에 대한 정보가 들어있는 단일 벡터로 변환
 - 그 벡터를 n 번 반복 (n 은 출력 시퀀스의 timestep의 수)
 - 그리고 이 일정한 시퀀스를 타겟 시퀀스로 바꾸기 위해 LSTM 디코더를 실행

신경망 설계

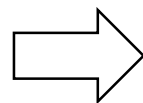


전력 데이터

- GEFCom2014 예측 대회에서 사용된 데이터
- 2012년과 2014년 사이의 3 년간의 시간별 전기 부하 및 온도 값으로 구성

timestamp	load	temp
2012-01-01 00:00:00	2,698.00	32.00
2012-01-01 01:00:00	2,558.00	32.67
2012-01-01 02:00:00	2,444.00	30.00
2012-01-01 03:00:00	2,402.00	31.00
2012-01-01 04:00:00	2,403.00	32.00

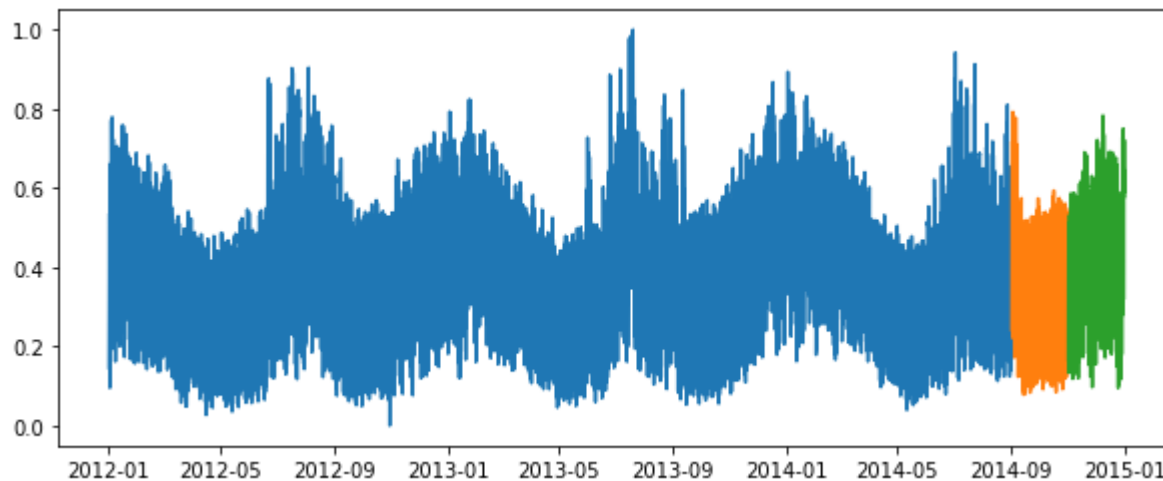
스케일링



timestamp	load	temp
2012-01-01 00:00:00	0.22	0.42
2012-01-01 01:00:00	0.18	0.43
2012-01-01 02:00:00	0.14	0.40
2012-01-01 03:00:00	0.13	0.41
2012-01-01 04:00:00	0.13	0.42

train, test, validation 데이터로 분리

구분	범위
train	2012-01-01 00:00:00 - 2014-08-31 00:00:00
validation	2014-09-01 00:00:00 - 2014-10-31 00:00:00
test	2014-11-01 00:00:00 - 2014-12-31 00:00:00



load 데이터에 대한 미래 시간 데이터 컬럼 추가 (미래 3시간)

```
train_load_shifted = train[['load']].copy()
train_load_shifted['y_t+1'] = train_load_shifted['load'].shift(-1,freq='H')
train_load_shifted['y_t+2'] = train_load_shifted['load'].shift(-2,freq='H')
train_load_shifted['y_t+3'] = train_load_shifted['load'].shift(-3,freq='H')
```

	load	y_t+1	y_t+2	y_t+3
timestamp				
2012-01-01 00:00:00	0.22	0.18	0.14	0.13
2012-01-01 01:00:00	0.18	0.14	0.13	0.13
2012-01-01 02:00:00	0.14	0.13	0.13	0.15
2012-01-01 03:00:00	0.13	0.13	0.15	0.18
2012-01-01 04:00:00	0.13	0.15	0.18	0.23

load 데이터에 대한 과거 시간 데이터 컬럼 추가 (과거 6시간)

```
train_load_shifted['load_t-5'] = train_load_shifted['load'].shift(5,freq='H')
train_load_shifted['load_t-4'] = train_load_shifted['load'].shift(4,freq='H')
train_load_shifted['load_t-3'] = train_load_shifted['load'].shift(3,freq='H')
train_load_shifted['load_t-2'] = train_load_shifted['load'].shift(2,freq='H')
train_load_shifted['load_t-1'] = train_load_shifted['load'].shift(1,freq='H')
train_load_shifted['load_t-0'] = train_load_shifted['load'].shift(0,freq='H')
```

	load	y_t+1	y_t+2	y_t+3	load_t-5	load_t-4	load_t-3	load_t-2	load_t-1	load_t-0
timestamp										
2012-01-01 00:00:00	0.22	0.18	0.14	0.13	nan	nan	nan	nan	nan	0.22
2012-01-01 01:00:00	0.18	0.14	0.13	0.13	nan	nan	nan	nan	0.22	0.18
2012-01-01 02:00:00	0.14	0.13	0.13	0.15	nan	nan	nan	0.22	0.18	0.14
2012-01-01 03:00:00	0.13	0.13	0.15	0.18	nan	nan	0.22	0.18	0.14	0.13
2012-01-01 04:00:00	0.13	0.15	0.18	0.23	nan	0.22	0.18	0.14	0.13	0.13

temp 데이터에 대한 과거 시간 데이터 컬럼 추가 (과거 6시간)

```
train_temp_shifted = train[['temp']].copy()
train_temp_shifted['temp_t-5'] = train_temp_shifted['temp'].shift(5,freq='H')
train_temp_shifted['temp_t-4'] = train_temp_shifted['temp'].shift(4,freq='H')
train_temp_shifted['temp_t-3'] = train_temp_shifted['temp'].shift(3,freq='H')
train_temp_shifted['temp_t-2'] = train_temp_shifted['temp'].shift(2,freq='H')
train_temp_shifted['temp_t-1'] = train_temp_shifted['temp'].shift(1,freq='H')
train_temp_shifted['temp_t-0'] = train_temp_shifted['temp'].shift(0,freq='H')
```

	temp	temp_t-5	temp_t-4	temp_t-3	temp_t-2	temp_t-1	temp_t-0
timestamp							
2012-01-01 00:00:00	0.42	nan	nan	nan	nan	nan	0.42
2012-01-01 01:00:00	0.43	nan	nan	nan	nan	0.42	0.43
2012-01-01 02:00:00	0.40	nan	nan	nan	0.42	0.43	0.40
2012-01-01 03:00:00	0.41	nan	nan	0.42	0.43	0.40	0.41
2012-01-01 04:00:00	0.42	nan	0.42	0.43	0.40	0.41	0.42

데이터 병합 및 결측치 제거, load/temp 열 삭제

load와 temp 데이터 병합

```
train_shifted = pd.concat([train_load_shifted, train_temp_shifted], axis=1)
```

결측치 제거

how -> any : 행에서 하나의 열이 결측치면 행 삭제

```
train_shifted = train_shifted.dropna(how='any')
```

load, temp 컬럼 삭제

```
train_shifted.drop(['load', 'temp'], inplace=True, axis=1)
```

	y_t+1	y_t+2	y_t+3	load_t-5	load_t-4	load_t-3	load_t-2	load_t-1	load_t-0	temp_t-5	temp_t-4	temp_t-3	temp_t-2	temp_t-1	temp_t-0
timestamp															
2012-01-01 05:00:00	0.18	0.23	0.29	0.22	0.18	0.14	0.13	0.13	0.15	0.42	0.43	0.40	0.41	0.42	0.41
2012-01-01 06:00:00	0.23	0.29	0.35	0.18	0.14	0.13	0.13	0.15	0.18	0.43	0.40	0.41	0.42	0.41	0.40
2012-01-01 07:00:00	0.29	0.35	0.37	0.14	0.13	0.13	0.15	0.18	0.23	0.40	0.41	0.42	0.41	0.40	0.39
2012-01-01 08:00:00	0.35	0.37	0.37	0.13	0.13	0.15	0.18	0.23	0.29	0.41	0.42	0.41	0.40	0.39	0.39
2012-01-01 09:00:00	0.37	0.37	0.37	0.13	0.15	0.18	0.23	0.29	0.35	0.42	0.41	0.40	0.39	0.39	0.43

**validation과 test 데이터도
동일하게 변경해보자**

load_n과 temp_n은 각각 X로
Y_n은 y로 분리해보자
(train, validation, test)

timestamp	load_t-5	load_t-4	load_t-3	load_t-2	load_t-1	load_t-0	temp_t-5	temp_t-4	temp_t-3	temp_t-2	temp_t-1	temp_t-0
2012-01-01 05:00:00	0.22	0.18	0.14	0.13	0.13	0.15	0.42	0.43	0.40	0.41	0.42	0.41
2012-01-01 06:00:00	0.18	0.14	0.13	0.13	0.15	0.18	0.43	0.40	0.41	0.42	0.41	0.40
2012-01-01 07:00:00	0.14	0.13	0.13	0.15	0.18	0.23	0.40	0.41	0.42	0.41	0.40	0.39
2012-01-01 08:00:00	0.13	0.13	0.15	0.18	0.23	0.29	0.41	0.42	0.41	0.40	0.39	0.39
2012-01-01 09:00:00	0.13	0.15	0.18	0.23	0.29	0.35	0.42	0.41	0.40	0.39	0.39	0.43

timestamp	y_t+1	y_t+2	y_t+3
2012-01-01 05:00:00	0.18	0.23	0.29
2012-01-01 06:00:00	0.23	0.29	0.35
2012-01-01 07:00:00	0.29	0.35	0.37
2012-01-01 08:00:00	0.35	0.37	0.37
2012-01-01 09:00:00	0.37	0.37	0.37

X를 timestep(6)과 feature(2)로 분리

```
X_train = X_train.to_numpy().reshape(X_train.shape[0], T, 2)
```

```
X_train.shape, y_train.shape
```

((23368, 6, 2), (23368, 3))

**validation과 test 데이터도
동일하게 변경해보자**

신경망 설계

```
LATENT_DIM = 5
```

```
autoencoder_F = Sequential()
```

```
# 인코더 부분
```

```
autoencoder_F.add(LSTM(LATENT_DIM, input_shape=(T, 2),  
activation="relu"))
```


신경망 설계

```
# 디코더 부분
# 벡터를 3번 반복 (디코더의 개수) - 예측할 미래의 개수
autoencoder_F.add(RepeatVector(HORIZON))
# return_sequences : 각 출력에서 시퀀스 출력 여부
# false이면 마지막 출력에서만 시퀀스가 출력됨
autoencoder_F.add(LSTM(LATENT_DIM, return_sequences=True,
activation="relu"))
# 하나의 노드마다 출력을 하나씩 뽑는다
autoencoder_F.add(TimeDistributed(Dense(1)))

autoencoder_F.add(Flatten())
```

학습

```
autoencoder_F.compile(optimizer='adam', loss='mse')

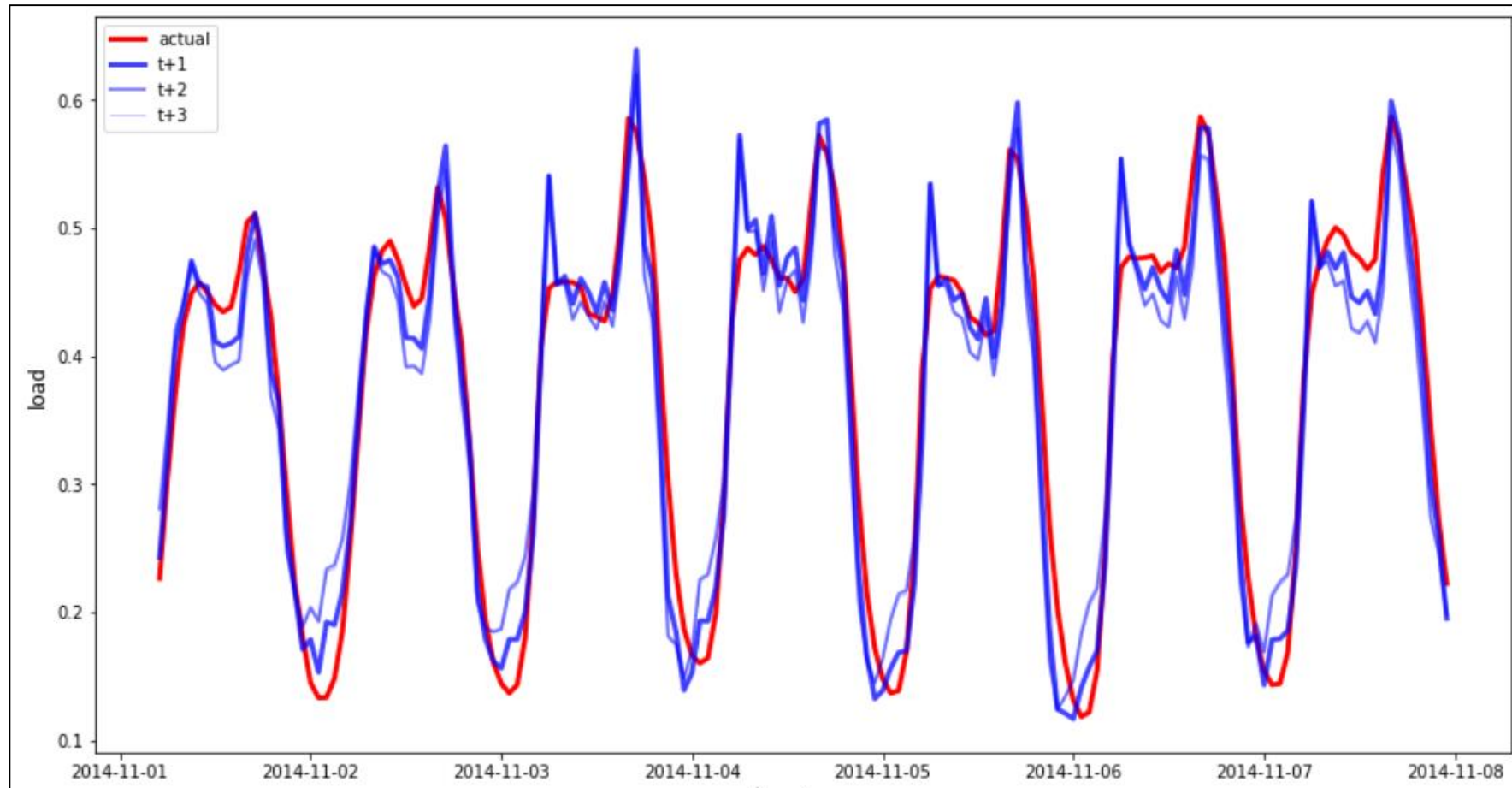
earlystop = EarlyStopping(monitor='val_loss', min_delta=0,
patience=5)

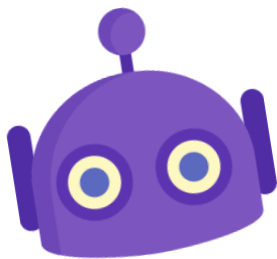
autoencoder_F.fit(X_train, y_train, batch_size=32, epochs=10,
validation_data=(X_valid, y_valid),
callbacks=[earlystop], verbose=1)
```

예측 결과

	timestamp	h	prediction	actual
0	2014-11-01 05:00:00	t+1	0.31	0.23
1	2014-11-01 06:00:00	t+1	0.34	0.31
2	2014-11-01 07:00:00	t+1	0.39	0.37
3	2014-11-01 08:00:00	t+1	0.41	0.42
4	2014-11-01 09:00:00	t+1	0.47	0.45

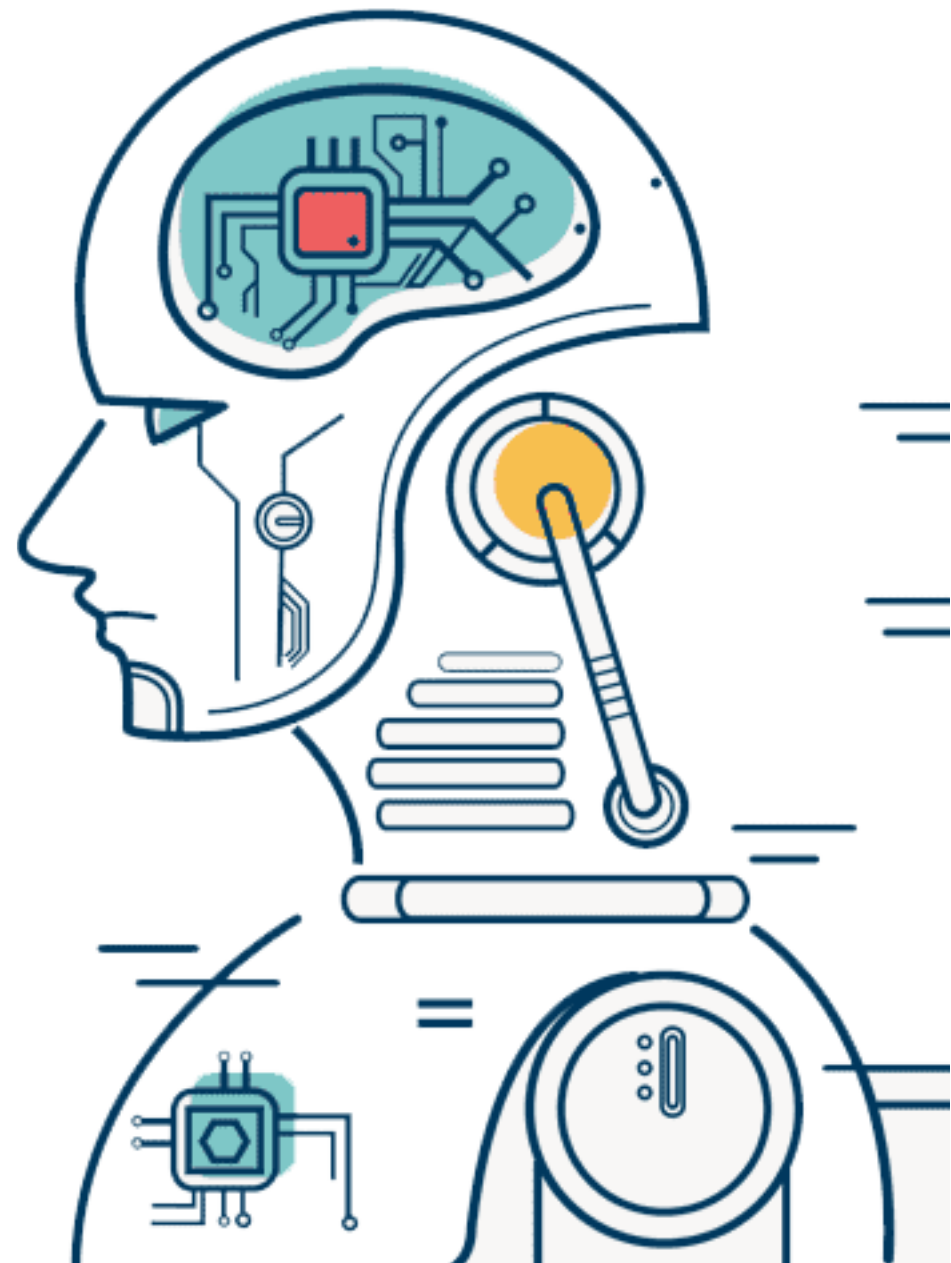
예측 결과





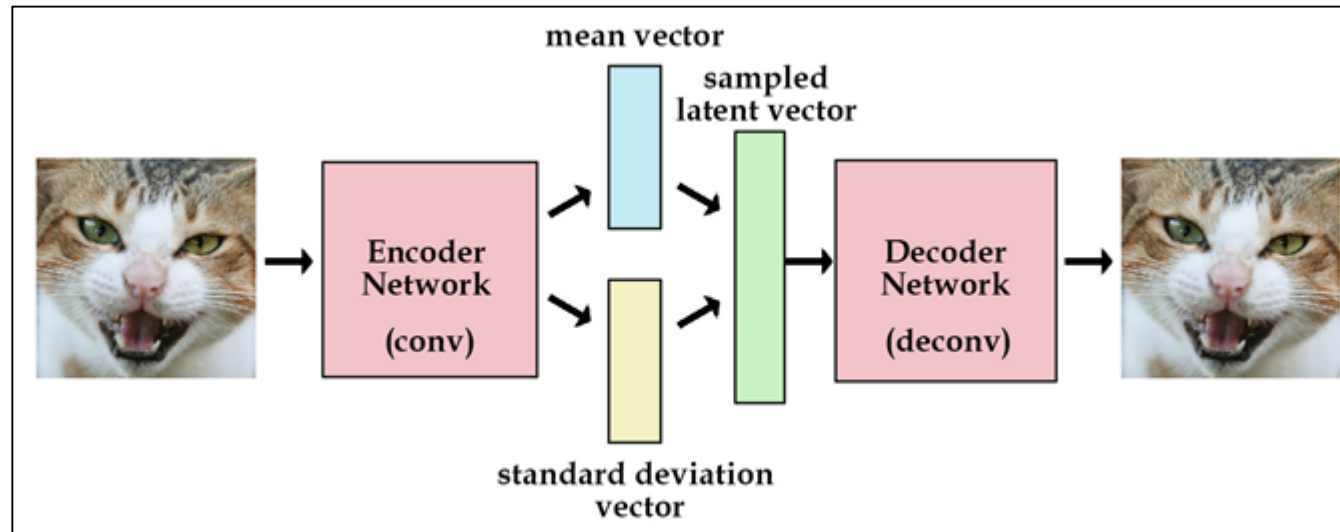
Autoencoder (AE)

Variational Autoencoder (VAE)



Variational Autoencoder (VAE)

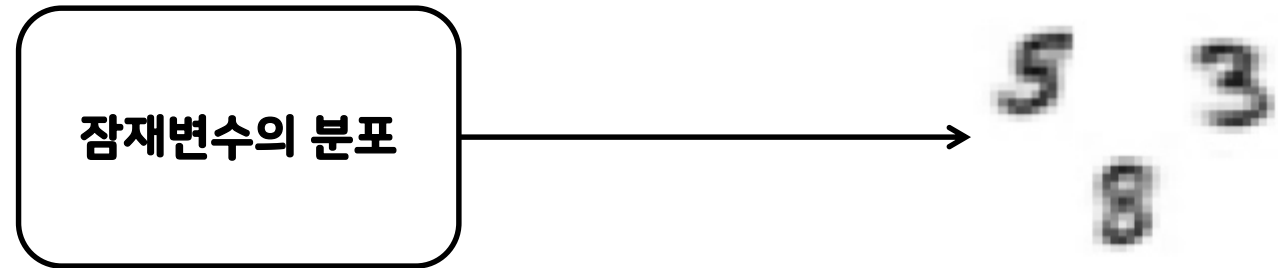
- 입력 데이터에 대한 잠재변수(latent variable) 모델을 학습하는 autoencoder
 - 생성 모델 (Generative Model)
- 잠재변수 z : 고양이를 구분하기 위한 특성 (털, 색깔, 눈, 모양, 이빨 개수 등의 추상화된 특성)
- Encoder 신경망 : 입력 데이터를 추상화하여 잠재적인 특징을 추출하는 역할
- Decoder 신경망 : 잠재적인 특징을 바탕으로 원 데이터를 복원하는 역할



- 신경망에게 데이터를 모델링하는 **확률 분포의 매개 변수를 학습** → 이 분포에서 점(points)를 샘플링하면, 새로운 입력 데이터의 샘플 또한 생성할 수 있음
- (1) encoder 네트워크는 입력 샘플 x 를 잠재공간(latent space)에서 두 개의 매개 변수로 변환 (z_mean 과 z_log_sigma)
 - (2) $z = z_mean + \exp(z_log_sigma) * \epsilon$ 을 통해 데이터를 생성한다고 가정한 잠재정규분포 (latent normal distribution)에서 유사한 점 z 를 무작위로 샘플링 (ϵ : 임의의 정규텐서 (normal tensor))
 - (3) decoder 네트워크는 이러한 잠재 공간의 점을 원래의 입력 데이터로 다시 매핑

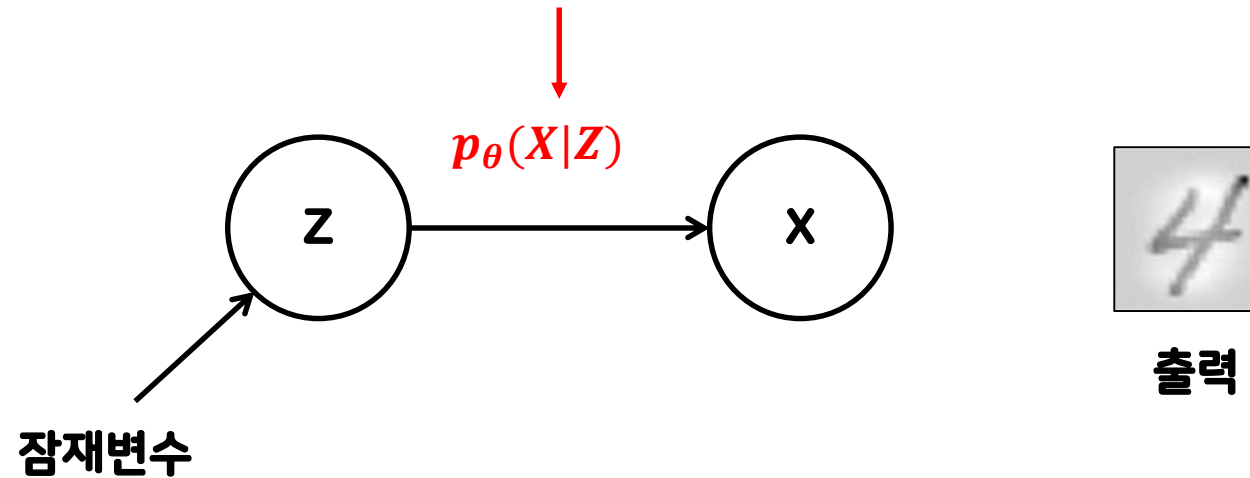
예를 들어 숫자의 잠재적인 의미를 생각하면

- 숫자(3인가 5인가)와 필체를 나타내는 잠재변수 (각도, aspect ratio 등)으로부터 숫자를 생성할 수 있음



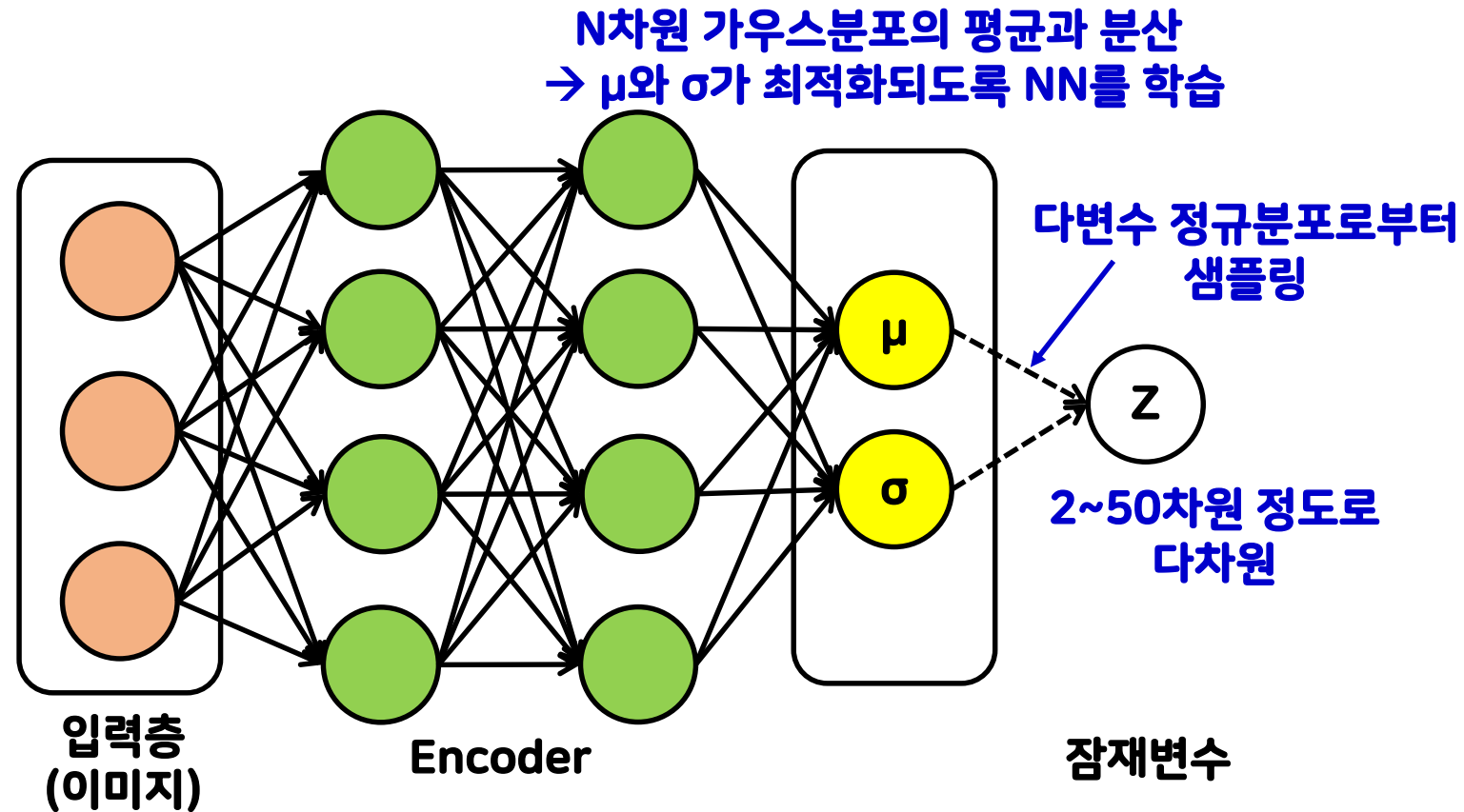
생성 모델의 최적화 전제

Z를 바탕으로 X가 생성되는 확률분포

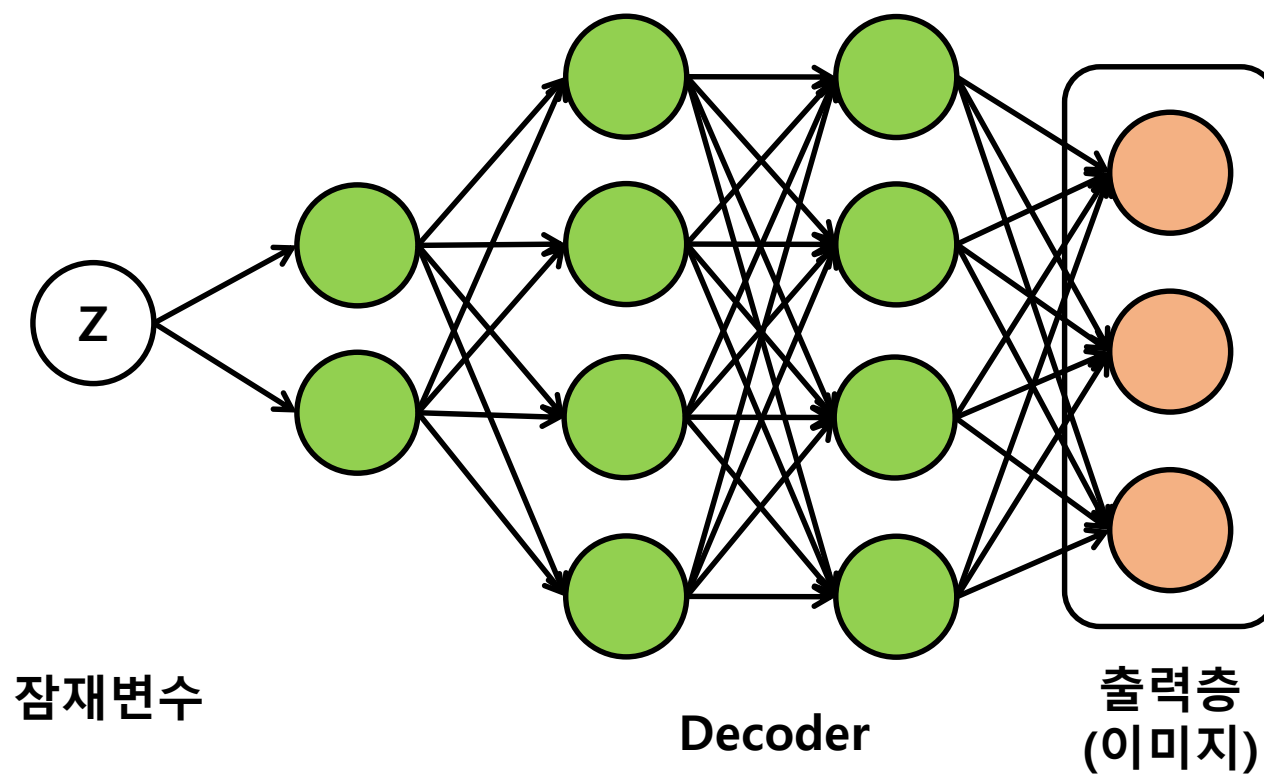


Z는 저차원인데 반해 X는 고차원이므로
입력(잠재변수)에 대응하는 답이 불명확함

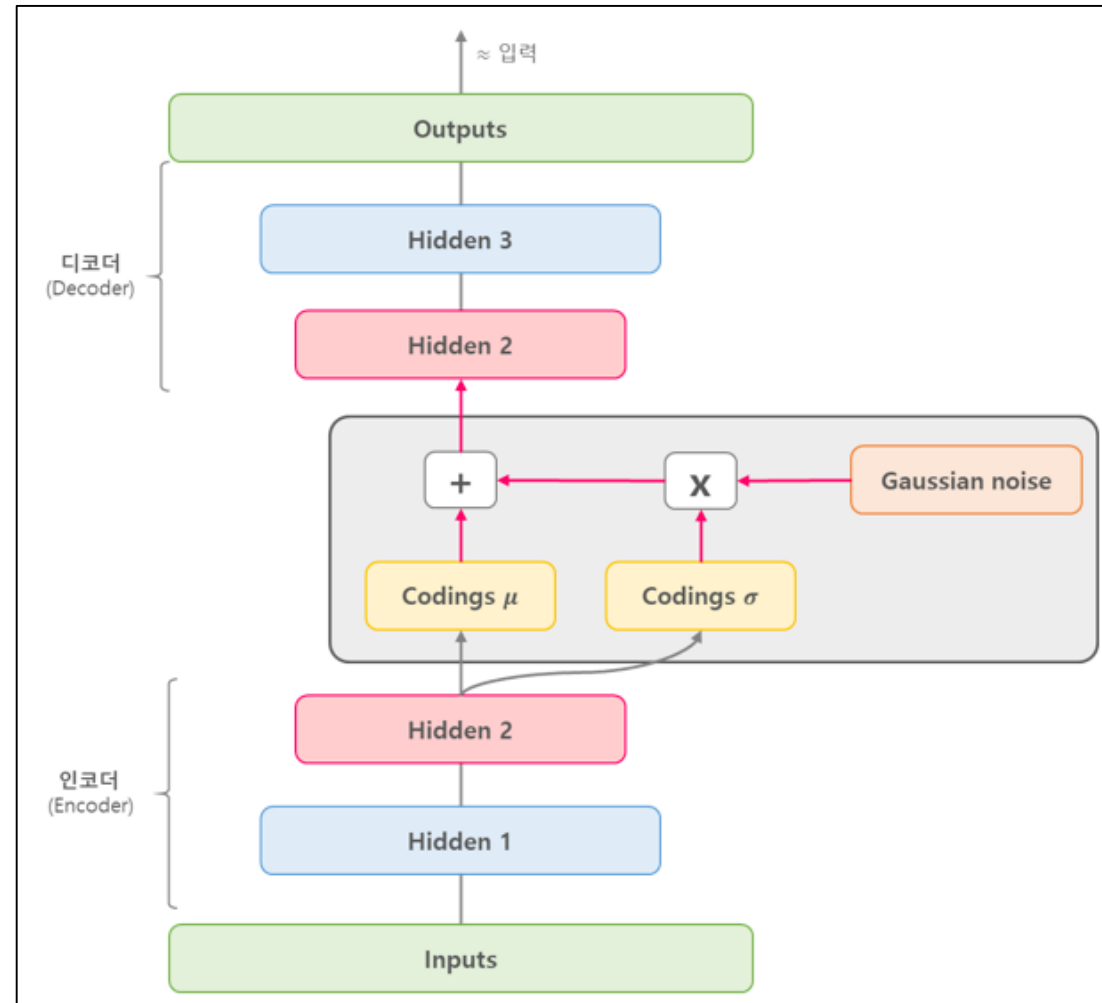
잠재 변수를 정규 분포로 가정



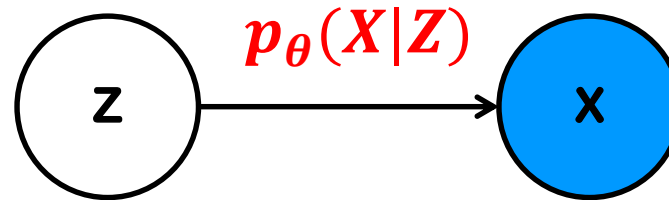
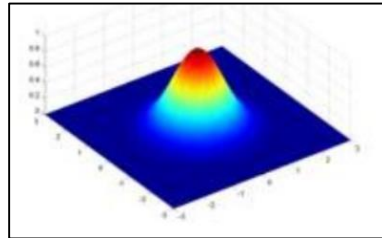
Decoder



Variational Autoencoder (VAE)



잠재변수(z)를 다차원 정규분포로 가정 $\rightarrow p(z)$



잠재변수 Z 로 부터 이미지 X 를 생성
(θ 는 파라미터)



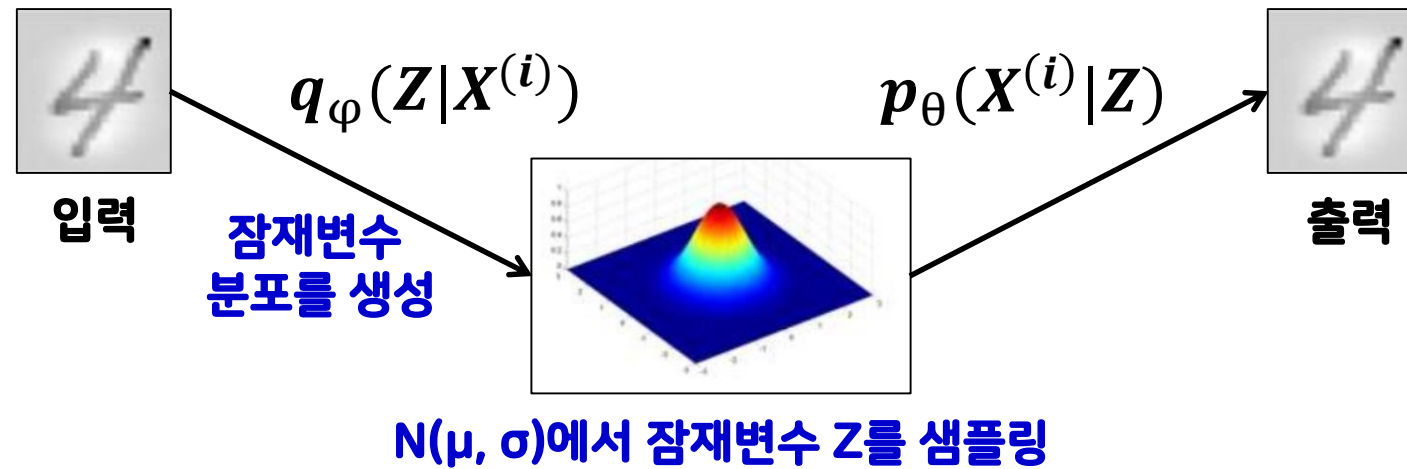
출력

- **Variational inference** : 이상적인 확률분포를 모르지만, 이를 추정하기 위해서 다루기 쉬운 분포 (approximation class, 대표적으로 Gaussian distribution)를 가정하고 이 확률분포의 모수를 바꿔가며, 이상적인 확률분포에 근사하게 만들어 그 확률분포를 대신 사용하는 것



Variational Autoencoder (VAE)

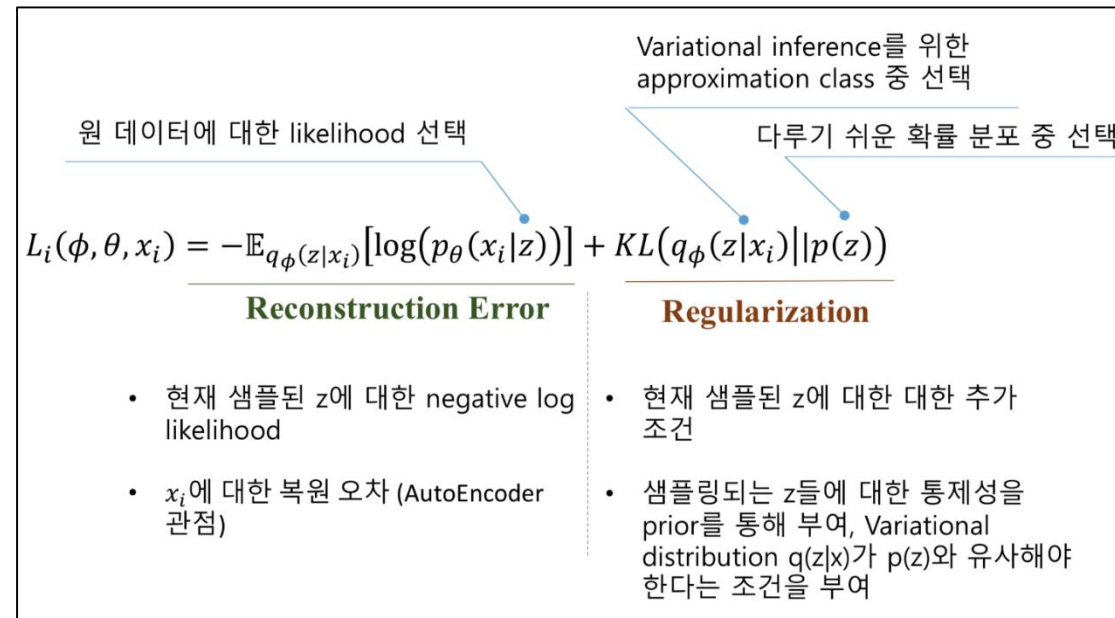
- 입력으로부터 잠재변수 분포를 생성
- 잠재변수로부터 샘플링 → 입력에 가까운 출력을 생성



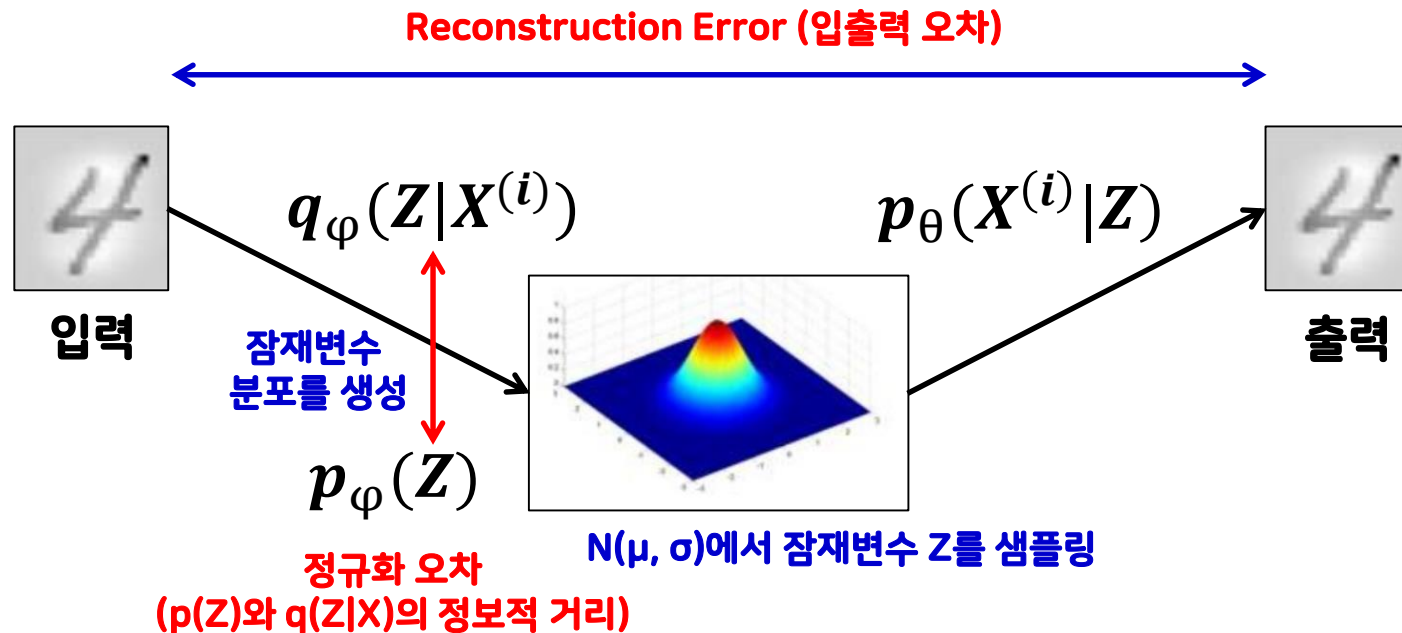
- 모델의 매개 변수는 두 가지 손실 함수를 통해 훈련

(1) 디코딩된 샘플을 초기 입력과 일치하도록하는 재구성 손실

(2) regularization term처럼 작동하는 잠재 분포(latent distribution)와 사전 분포(prior distribution) 간의 **KL 발산** (이상적인 분포에 근사하는 다른 분포를 사용해 샘플링하는 경우에 발생할 수 있는 정보 엔트로피 차이)를 계산)

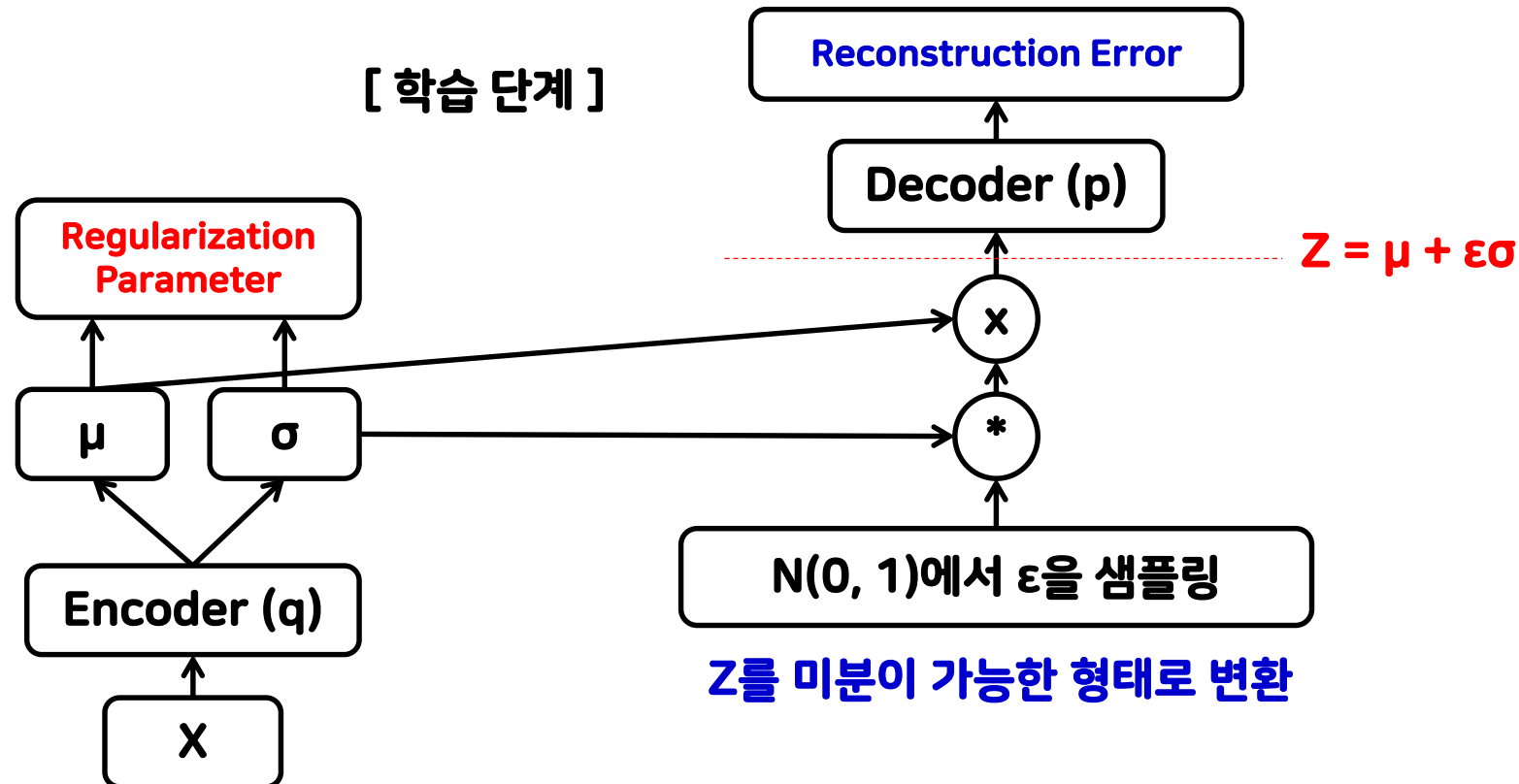


최적화 함수

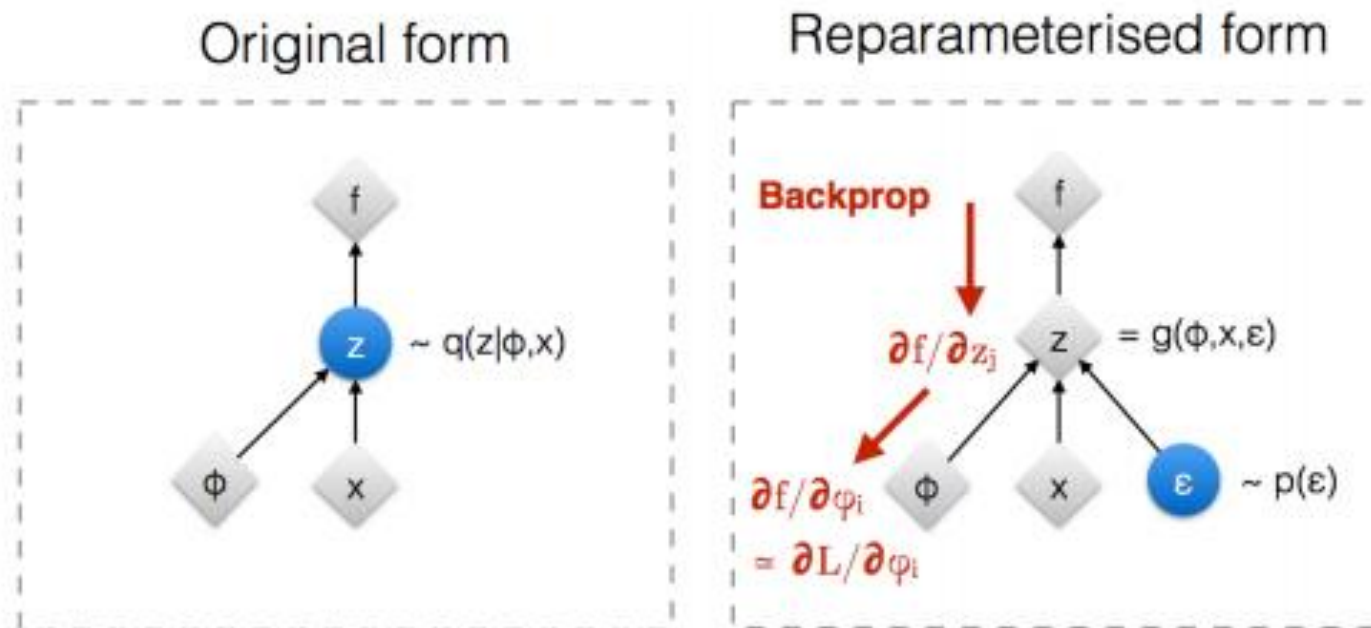


Variational Autoencoder (VAE)

- **reparameterization trick** : 미분 불가능한 평균과 분산으로 샘플링된 가우시안 벡터를 미분 가능한 가우시안 샘플링 메소드로 만드는 것 → **보조 잡음 변수 ϵ 를 사용**



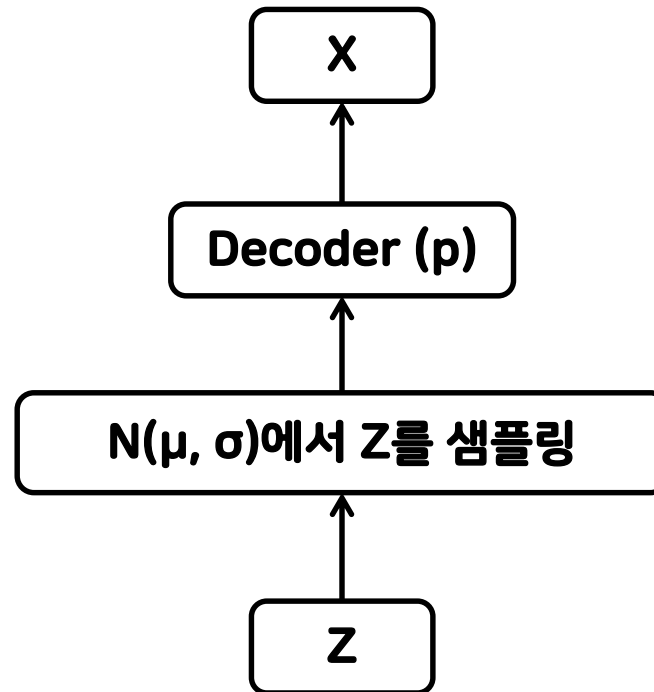
Reparameterization trick



◊ : Deterministic node
● : Random node

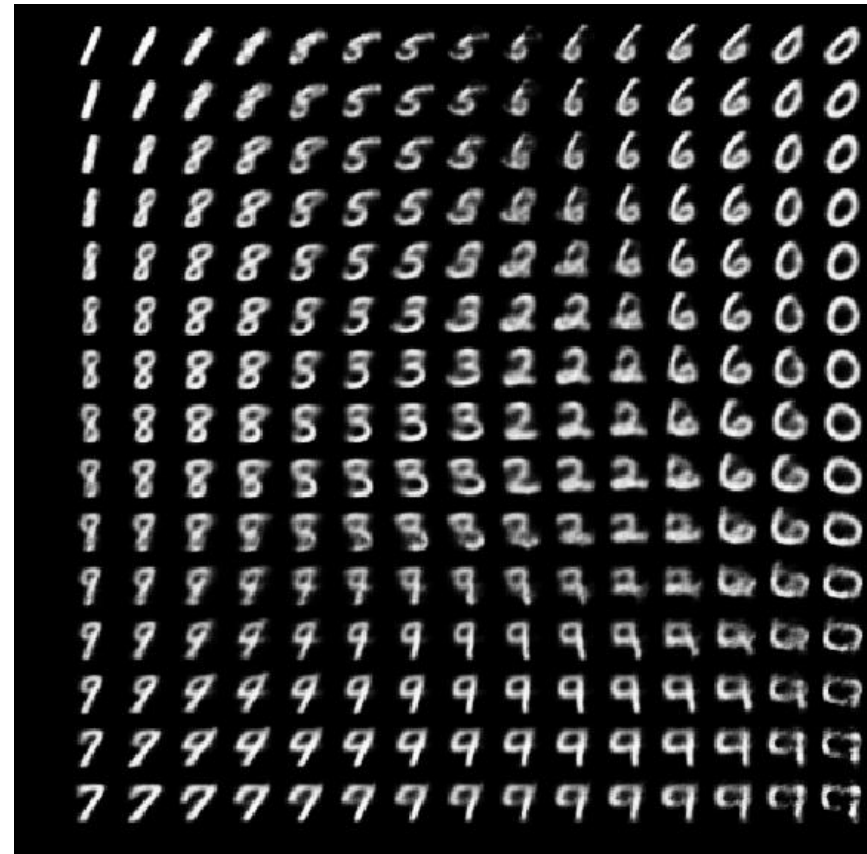
[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

활용 단계



- VAE는 GAN에 비해 학습이 안정적인 편 → 손실함수에서 확인할 수 있듯이 reconstruction error와 같이 평가 기준이 명확하기 때문
- 데이터뿐만 아니라 데이터에 내재한 잠재변수 z 도 함께 학습할 수 있다는 장점 (feature learning)
- 출력이 선명하지 않고 평균값 형태로 표시되는 문제
- reparameterization trick이 모든 경우에 적용되지 않는 문제 등

실행 결과



VAE를 이용한 숫자 생성

VAE를 이용한 얼굴 이미지 생성

- CelebA (CelebFaces Attributes) 데이터 셋을 이용
- <http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html> (유명인사의 컬러 이미지 20만개, 라벨 포함)

안경



뱅 스타일



끝이 뾰족한 코



계란형 얼굴



모자를 쓴



웨이브 머리



코밑 수염



미소



- <https://ratsgo.github.io/generative%20model/2018/01/28/VAEs/>
- **Conditional VAE** : 지도학습이 가능하도록 y 값을 추가한 모델
- **Adversarial Autoencoder (AAE)** : VAE에 GAN를 덧입힌 구조
- **Sketch RNN** : VAE에 RNN 구조를 덧입힌 아키텍처

Adversarial Autoencoder (AAE)

<http://incredible.egloos.com/7473304>