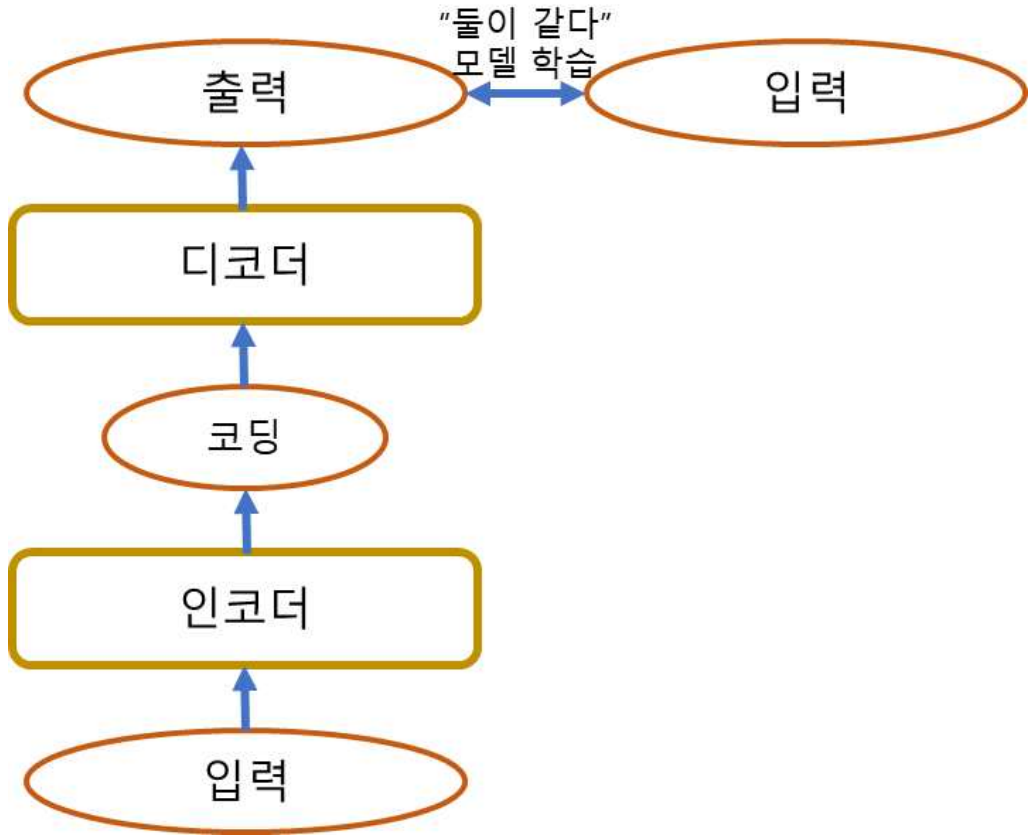
오토인코더

목표 출력이 없어도 입력만으로 구성된 훈련 데이터로 비지도 학습을 수행

- 특성 추출
- 비지도 사전훈련
- 차원 축소
- 생성 모델
- 이상치 탐지



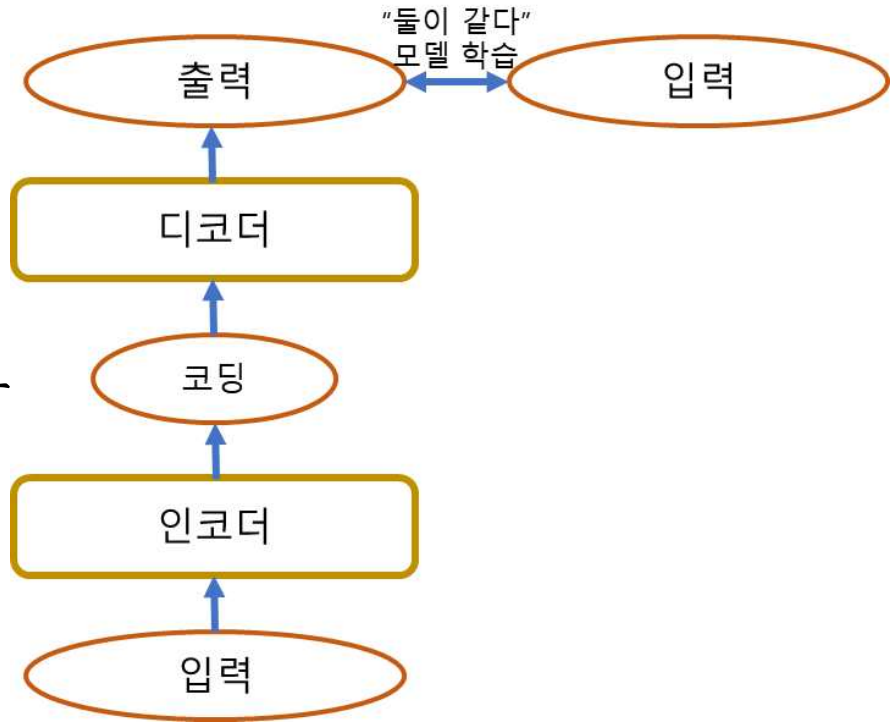
인코더: 입력보다 작은 차원으로 데이터의 성향(패턴) 파악

→ 입력 차원 > 코딩 차원: undercomplete

디코더: 파악한 데이터의 성향을 이용해 데이터 재구성

예) SVD 기법 없이 PCA 수행 가능

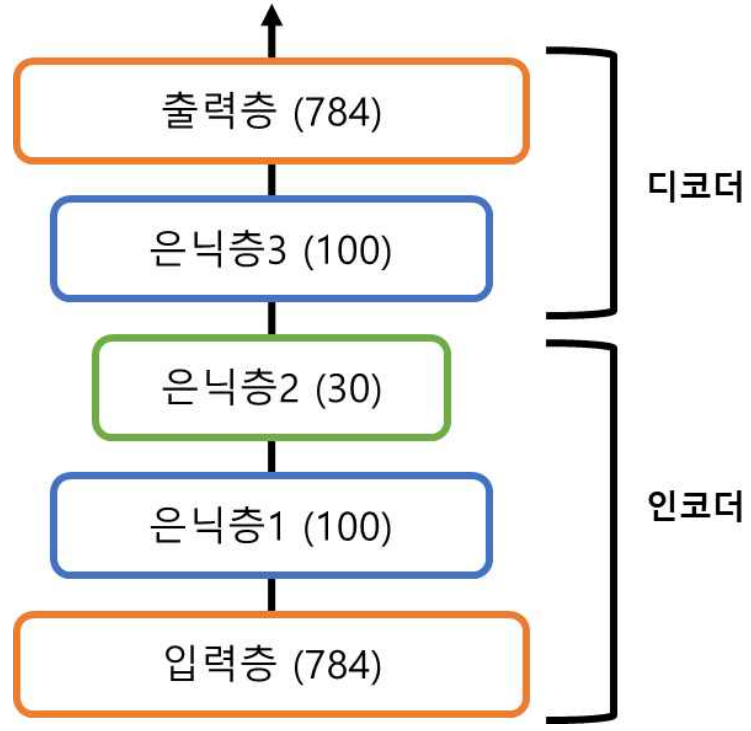
- 입력 3차원, 코딩 2차원 훈련하면, 분산이 가능한 많이 보존되는 최적의 평면 도출 가능



적층 오토인코더

인코더와 디코더에 은닉층 여러개

```
[3] # 인코더
    stacked_encoder = keras.models.Sequential(
        [keras.layers.Flatten(input_shape=[28, 28]), # (28, 28) → 784
         keras.layers.Dense(100, activation='selu'), # 784 → 100
         keras.layers.Dense(30, activation='selu')] # 100 → 30
    )

    # 디코더
    stacked_decoder = keras.models.Sequential(
        [keras.layers.Dense(100, activation='selu', input_shape=[30]), # 30 → 100
         keras.layers.Dense(28*28, activation='sigmoid'), # 100 → 784, 0~1 데이터
         keras.layers.Reshape([28, 28])] # 784 → (28, 28)
    )

    # 오토인코더
    stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
```



비용함수: 이진 크로스엔트로피

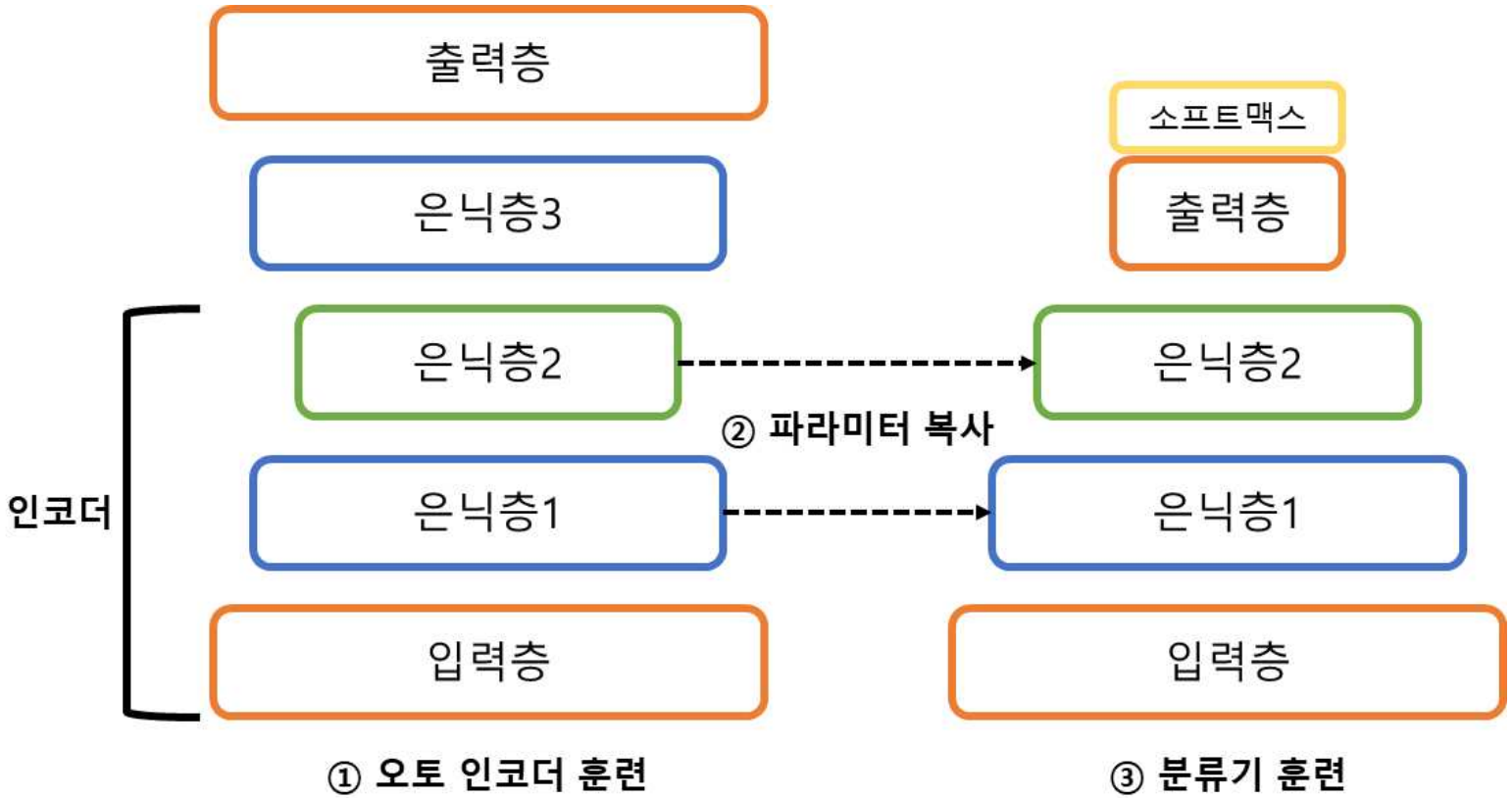
재구성 데이터 시각화

패션 MNIST 데이터 中

위: 원본 데이터 / 아래: 재구성 데이터

비지도 사전훈련

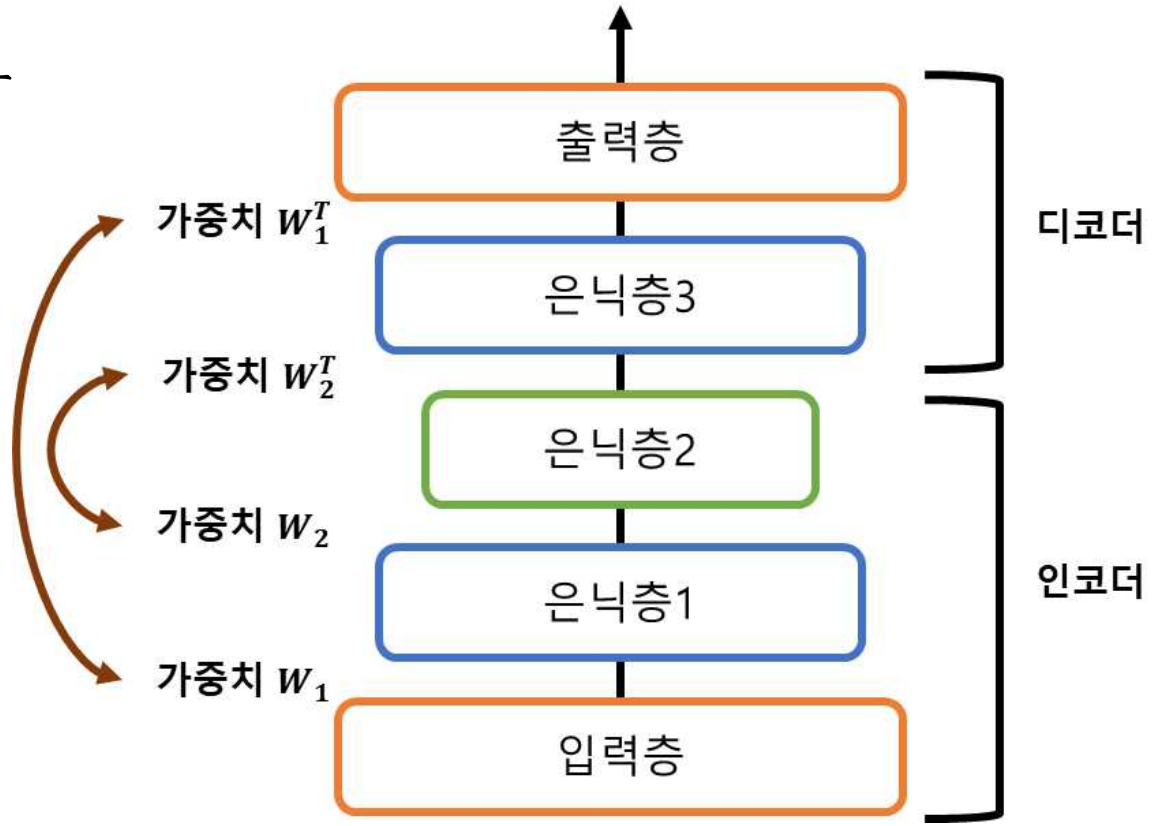
상황: 레이블된 데이터는 적고, 레이블 되지 않은 데이터는 많은 경우



가중치 복사

대칭이 되는 가중치 값을 같게 고정

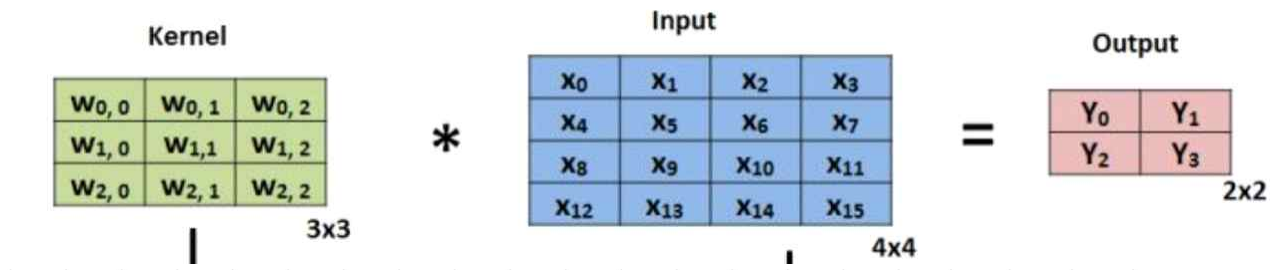
훈련해야할 가중치의 개수가 절반으로 줄어듦, 과대적합 방지

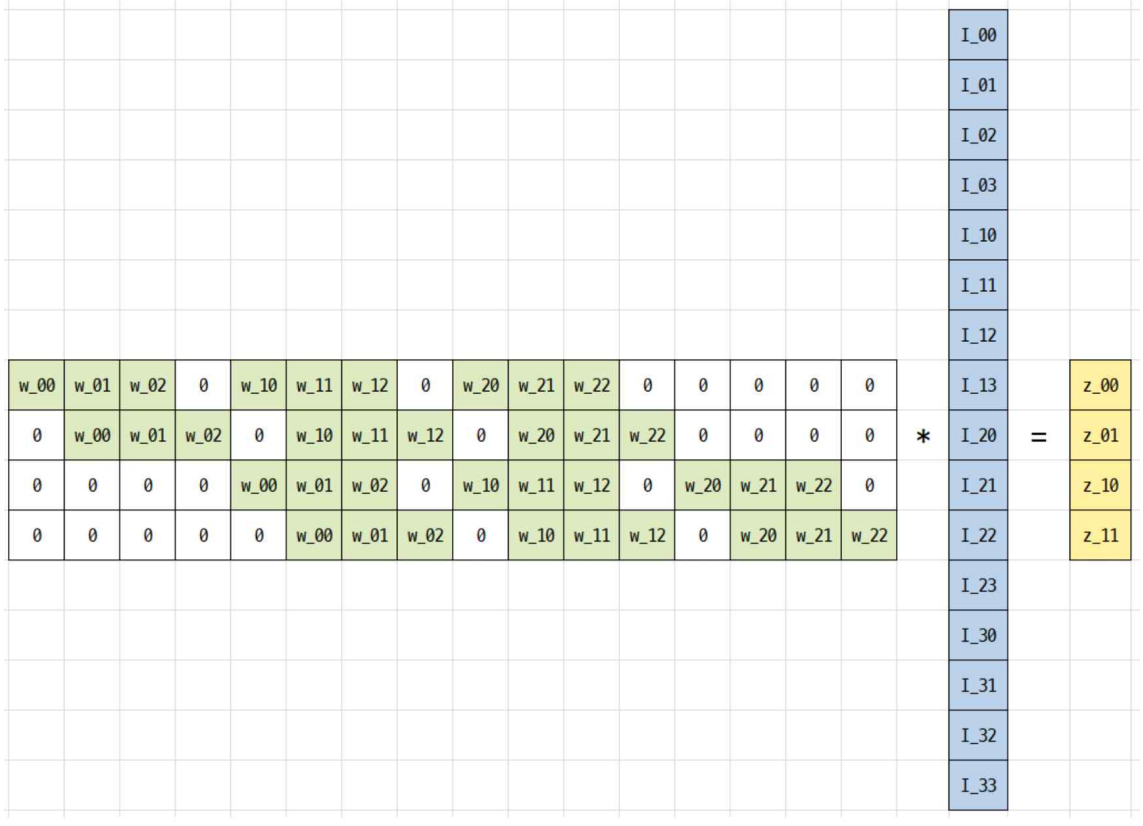


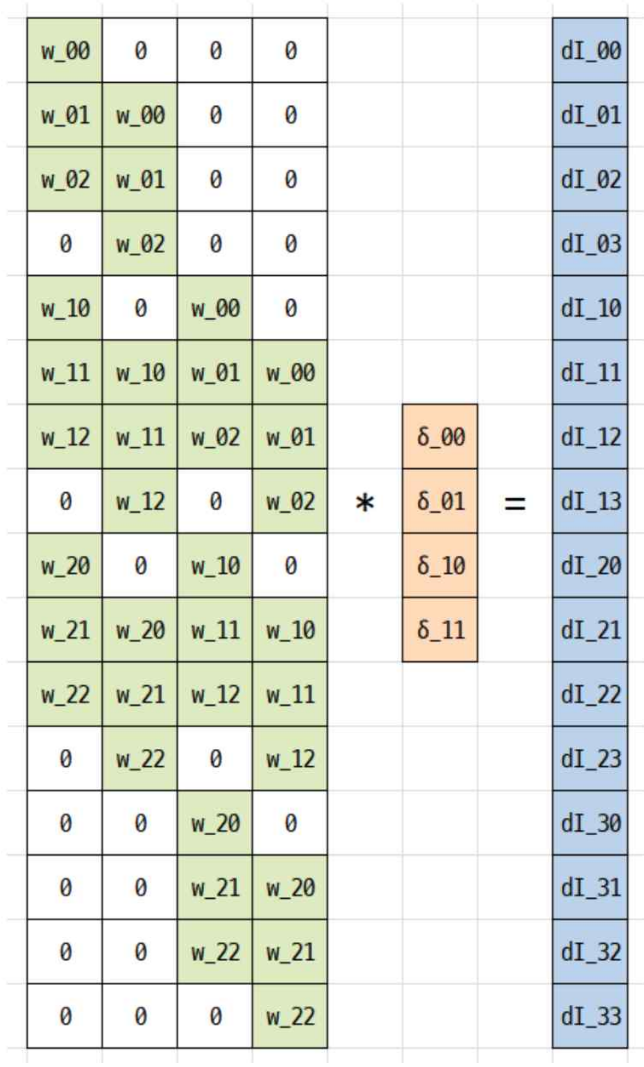
합성곱 오토인코더

```
# 인코더
conv_encoder = keras.models.Sequential(
    [keras.layers.Reshape([28, 28, 1], input_shape=[28, 28]),
     keras.layers.Conv2D(16, kernel_size=3, padding='same', activation='selu'),
     keras.layers.MaxPool2D(pool_size=2),
     keras.layers.Conv2D(32, kernel_size=3, padding='same', activation='selu'),
     keras.layers.MaxPool2D(pool_size=2),
     keras.layers.Conv2D(64, kernel_size=3, padding='same', activation='selu'),
     keras.layers.MaxPool2D(pool_size=2)]
)

# 디코더
conv_decoder = keras.models.Sequential(
    [keras.layers.Conv2DTranspose(32, kernel_size=3, strides=2, padding='valid', activation='selu', input_shape=[3, 3, 64]),
     keras.layers.Conv2DTranspose(16, kernel_size=3, strides=2, padding="same", activation='selu'),
     keras.layers.Conv2DTranspose(1, kernel_size=3, strides=2, padding="same",activation='sigmoid'),
     keras.layers.Reshape([28, 28])]
)

# 오토 인코더
conv_ae = keras.models.Sequential([conv_encoder, conv_decoder])
```

전치 합성곱


Kernel
Input
Output
w0, 0 w0, 1 w0, 2
w1, 0 w1,1 w1, 2
w2, 0 w2, 1 w2, 2
3x3
*
x0 x1 x2 x3
x4 x5 x6 x7
x8 x9 x10 x11
x12 x13 x14 x15
4x4
=
Y0 Y1
Y2 Y3
2x2


w_00 w_01 w_02 0 w_10 w_11 w_12 0 w_20 w_21 w_22 0 0 0 0 0
0 w_00 w_01 w_02 0 w_10 w_11 w_12 0 w_20 w_21 w_22 0 0 0 0
0 0 0 0 w_00 w_01 w_02 0 w_10 w_11 w_12 0 w_20 w_21 w_22 0
0 0 0 0 0 w_00 w_01 w_02 0 w_10 w_11 w_12 0 w_20 w_21 w_22
*
I_00 I_01 I_02 I_03 I_10 I_11 I_12 I_13 I_20 I_21 I_22 I_23 I_30 I_31 I_32 I_33
=
z_00 z_01 z_10 z_11


w_00 0 0 0
w_01 w_00 0 0
w_02 w_01 0 0
0 w_02 0 0
w_10 0 w_00 0
w_11 w_10 w_01 w_00
w_12 w_11 w_02 w_01
0 w_12 0 w_02
w_20 0 w_10 0
w_21 w_20 w_11 w_10
w_22 w_21 w_12 w_11
0 w_22 0 w_12
0 0 w_20 0
0 0 w_21 w_20
0 0 w_22 w_21
0 0 0 w_22
*
δ_00
δ_01
δ_10
δ_11
=
dI_00 dI_01 dI_02 dI_03 dI_10 dI_11 dI_12 dI_13 dI_20 dI_21 dI_22 dI_23 dI_30 dI_31 dI_32 dI_33

전치 합성곱

```
conv_decoder.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_transpose (Conv2DTran (None, 7, 7, 32)          18464
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 16)        4624
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 1)         145
_________________________________________________________________
reshape_1 (Reshape)          (None, 28, 28)            0
=================================================================
Total params: 23,233
Trainable params: 23,233
Non-trainable params: 0
_________________________________________________________________
```

순환 오토인코더

```
# 인코더
recurrent_encoder = keras.models.Sequential(
    [keras.layers.LSTM(100, return_sequences=True, input_shape=[None, 28]),
     keras.layers.LSTM(30)]
)

# 디코더
recurrent_decoder = keras.models.Sequential(
    [keras.layers.RepeatVector(28, input_shape=[30]), # 입력을 28번 반복
     keras.layers.LSTM(100, return_sequences=True),
     keras.layers.TimeDistributed(keras.layers.Dense(28, activation='sigmoid'))]
)

# 오토인코더
recurrent_ae = keras.models.Sequential([recurrent_encoder, recurrent_decoder])
```

```
recurrent_encoder.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, None, 100)         51600
_________________________________________________________________
lstm_3 (LSTM)                (None, 30)                15720
=================================================================
Total params: 67,320
Trainable params: 67,320
Non-trainable params: 0
_________________________________________________________________
```

```
recurrent_decoder.summary()
```

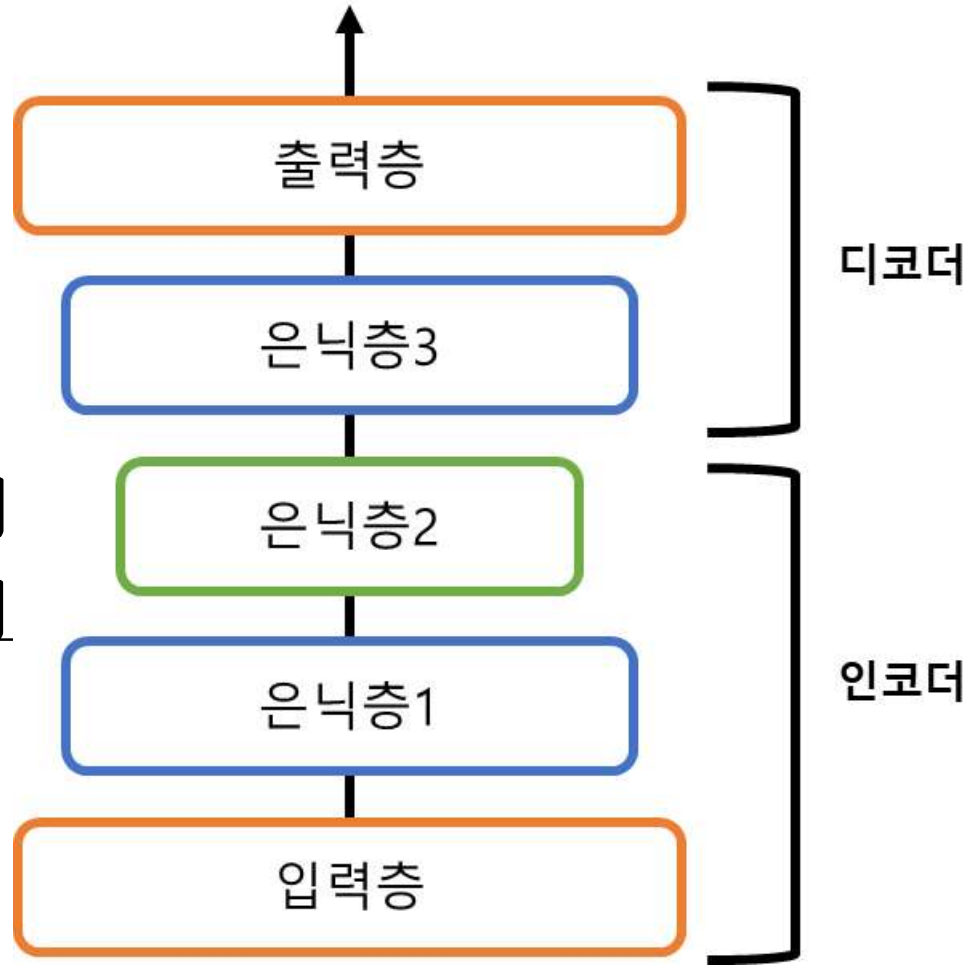
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
repeat_vector_1 (RepeatVecto (None, 28, 30)            0
_________________________________________________________________
lstm_4 (LSTM)                (None, 28, 100)           52400
_________________________________________________________________
time_distributed (TimeDistri (None, 28, 28)            2828
=================================================================
Total params: 55,228
Trainable params: 55,228
Non-trainable params: 0
_________________________________________________________________
```

Undercomplete Autoencoder

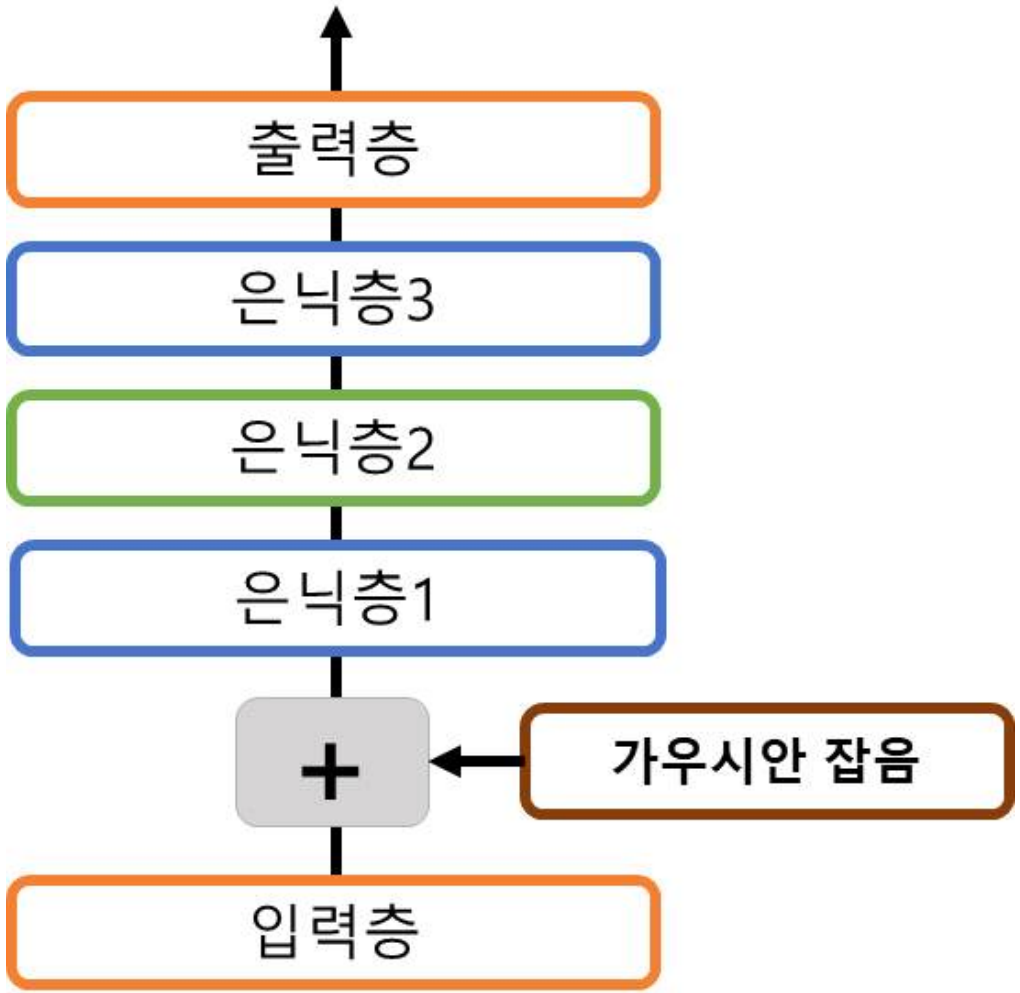
- 입력 차원 > 코딩 차원

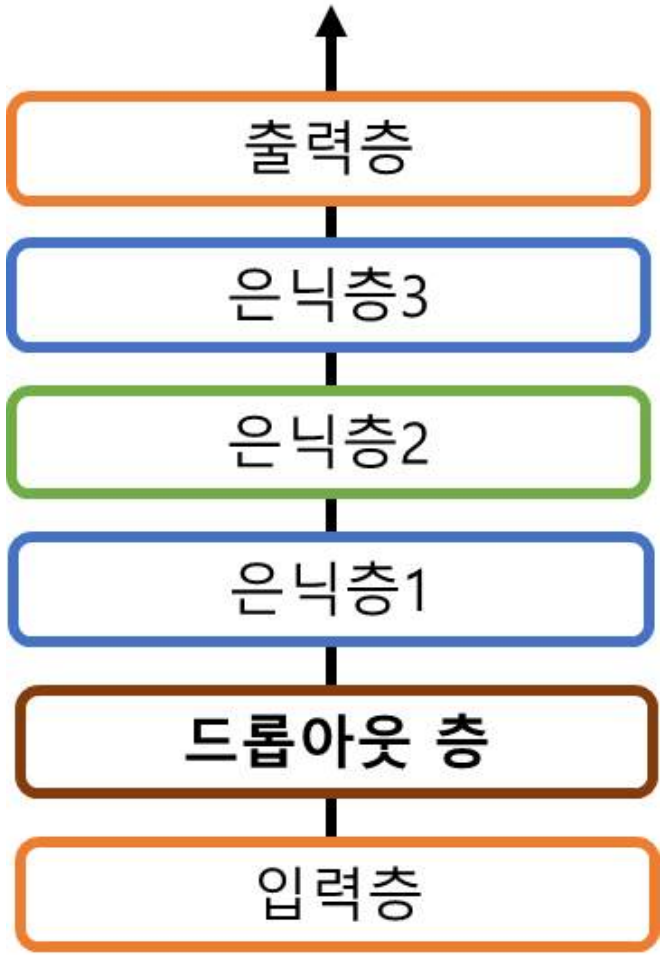
Overcomplete Autoencoder

- 입력 차원 < 코딩 차원
- 그냥 진행한다면 데이터의 성향 파악 의미 없음, 데이터의 성향을 파악하기 위한 장치 필요



잡음 제거 오토인코더 (Overcomplete)





희소 오토인코더

비용함수에 규제를 추가 → 코딩층에서 활성화되는 뉴런 수를 감소

코딩층에서 활성화되는 뉴런의 비율 (=희소 정도)의 목표를 직접 정할 수 있음

규제: 쿨백-라이블러 발산

$$D_{KL}(P\|Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)} = [-\sum_i P(i)\log Q(i)] - [-\sum_i P(i)\log P(i)]$$

목표 희소 정도: p / 실제 희소 정도: q

$$D_{KL}(p\|q) = p\log\frac{p}{q} + (1-p)\log\frac{1-p}{1-q}$$

희소 오토인코더 (KL 발산 규제)

```
K = keras.backend
kl_divergence = keras.losses.kullback_leibler_divergence

class KLDivergenceRegularizer(keras.regularizers.Regularizer):
  def __init__(self, weight, target=0.1):
    self.weight = weight  # 가중치
    self.target = target  # 목표 희소 정도
  def __call__(self, inputs):
    mean_activities = K.mean(inputs, axis=0) # 샘플 단위 평균
    return self.weight*(
        kl_divergence(self.target, mean_activities) +
        kl_divergence(1.-self.target, 1.-mean_activities)
    )
```

$$D_{KL}(p\|q) = p\log\frac{p}{q} + (1-p)\log\frac{1-p}{1-q}$$

희소 오토인코더 (KL 발산 규제)

```
# KL 발산 규제 설정: 가중치 0.05, 목표 희소 정도 0.1
kld_reg = KLDivergenceRegularizer(weight=0.05, target=0.1)

# 인코더
sparse_kl_encoder = keras.models.Sequential(
    [keras.layers.Flatten(input_shape=[28, 28]),
     keras.layers.Dense(100, activation='selu'),
     keras.layers.Dense(300, activation='sigmoid',
                        activity_regularizer=kld_reg)] # KL 발산 규제 적용
)

# 디코더
sparse_kl_decoder = keras.models.Sequential(
    [keras.layers.Dense(100, activation='selu', input_shape=[300]),
     keras.layers.Dense(28*28, activation='sigmoid'),
     keras.layers.Reshape([28, 28])]
)

# 오토인코더
sparse_kl_ae = keras.models.Sequential([sparse_kl_encoder, sparse_kl_decoder])
```

참고 자료

- 오헬리앙 제롱, *핸즈온 머신러닝 2판* (한빛미디어, 2020), 673-693.
- "합성곱 신경망에서 컨벌루션과 트랜스포즈드 컨벌루션의 관계," *Metamath*, 2020년 2월 29일 수정, 2021년 10월 2일 접속, https://metamath1.github.io/2019/05/09/transconv.html
- "딥러닝에서 사용되는 여러 유형의 Convolution 소개," *어쩐지 오늘은*, 2018년 2월 23일 수정, 2021년 10월 2일 접속, https://zzsza.github.io/data/2018/02/23/introduction-convolution/