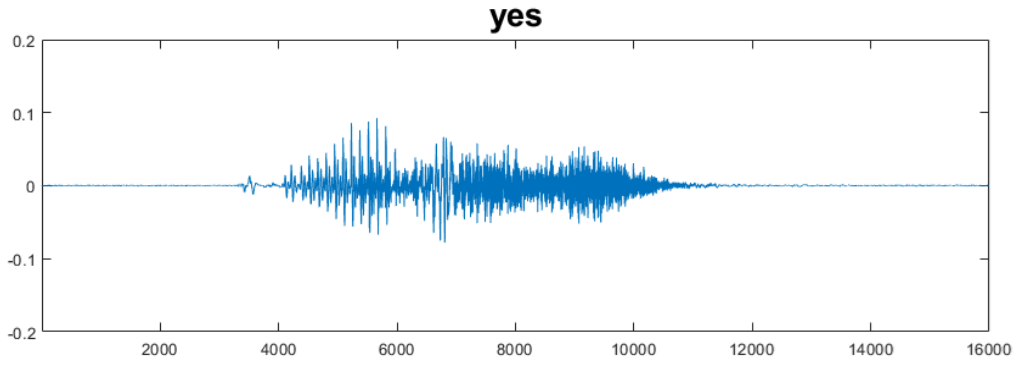
Time Series Anomaly Detection
LSTM + AE

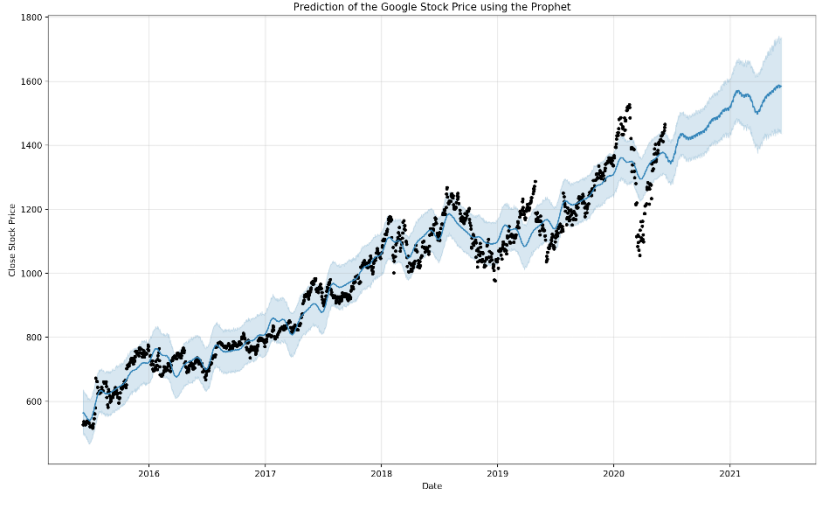
- 시계열 모델링을 위한 데이터구조
- RNN 및 LSTM의 기본 개념을 간략하게 살펴보고,
- LSTM+AE를 이용한 이상탐지를 수행해 봅니다.

Sequential Data

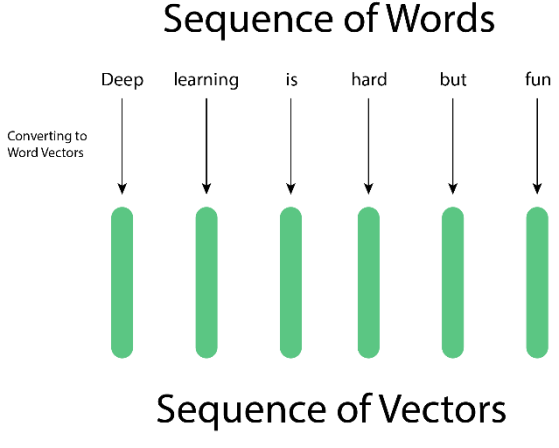
Voice



Stock Data



Sentence



데이터 전처리

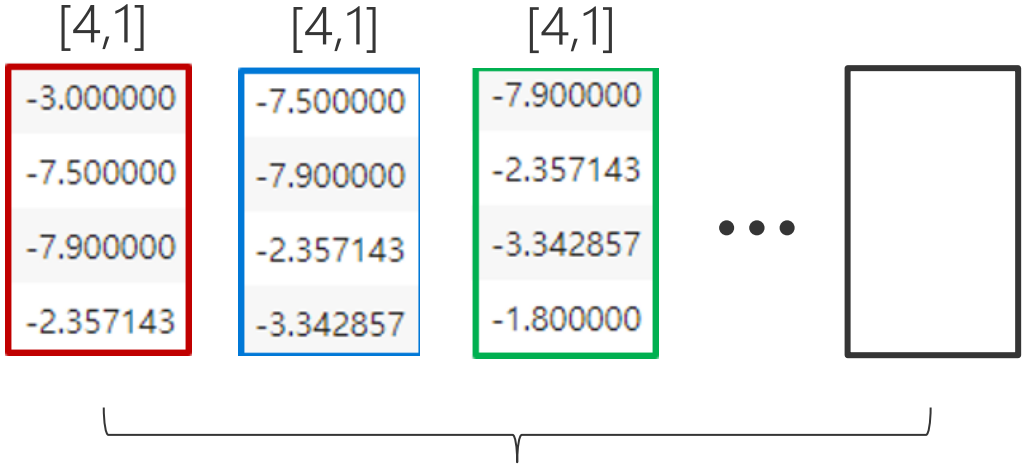
✓주간기온을 예측하는데 과거 어느 정도의 기간이면 적절할까?

	year	week	AvgTemp
0	2010	1	-3.000000
1	2010	2	-7.500000
2	2010	3	-7.900000
3	2010	4	-2.357143
4	2010	5	-3.342857
5	2010	6	-1.800000
6	2010	7	-0.314286
7	2010	8	-2.142857
8	2010	9	4.400000
9	2010	10	7.057143

데이터 전처리① : RNN을 위한 데이터구조

✓ 예를 들어, 과거 4주간의 데이터로 예측할 때, **X의 구조는?**

	year	week	AvgTemp
0	2010	1	-3.000000
1	2010	2	-7.500000
2	2010	3	-7.900000
3	2010	4	-2.357143
4	2010	5	-3.342857
5	2010	6	-1.800000
6	2010	7	-0.314286
7	2010	8	-2.142857
8	2010	9	4.400000
9	2010	10	7.057143



몇 개?

Total – timestep + 1 = #

3차원 데이터 : [#, 4, 1]

데이터 전처리① : RNN을 위한 데이터구조

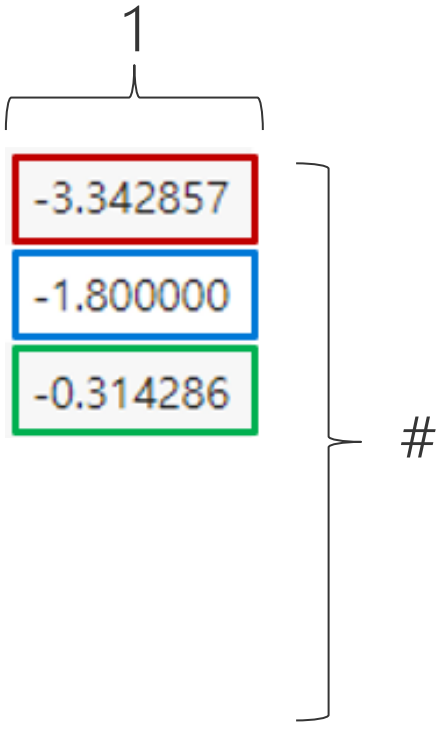
✓ 예를 들어, 과거 4주간의 데이터로 예측할 때, **Y의 구조는?**

	year	week	AvgTemp
0	2010	1	-3.000000
1	2010	2	-7.500000
2	2010	3	-7.900000
3	2010	4	-2.357143
4	2010	5	-3.342857
5	2010	6	-1.800000
6	2010	7	-0.314286
7	2010	8	-2.142857
8	2010	9	4.400000
9	2010	10	7.057143

0 1 2

	year	week	AvgTemp
0	2010	1	-3.000000
1	2010	2	-7.500000
2	2010	3	-7.900000
3	2010	4	-2.357143
4	2010	5	-3.342857
5	2010	6	-1.800000
6	2010	7	-0.314286
7	2010	8	-2.142857
8	2010	9	4.400000
9	2010	10	7.057143

0 1 2



데이터 전처리② : 스케일링

2차원 X	3차원 X ①	3차원 X ②
1) **데이터 분할** 2) **스케일링** - train set으로 Scaler 생성 - 전체 데이터에 적용	1) **데이터 분할** 2) **스케일링** - train set으로 Scaler 생성 - 전체 데이터에 적용 3) **2차원 → 3차원 변환**	1) **2차원 → 3차원 변환** 2) **데이터 분할** 3) **스케일링** - train set을 다시 2차원으로 변환 후 Scaler 생성 - 3차원 데이터에 적용
	3차원 변환 과정에서 샘플의 초기 데이터를 사용하지 못하게 됨.	변환 코드, 3차원 적용 코드 필요.

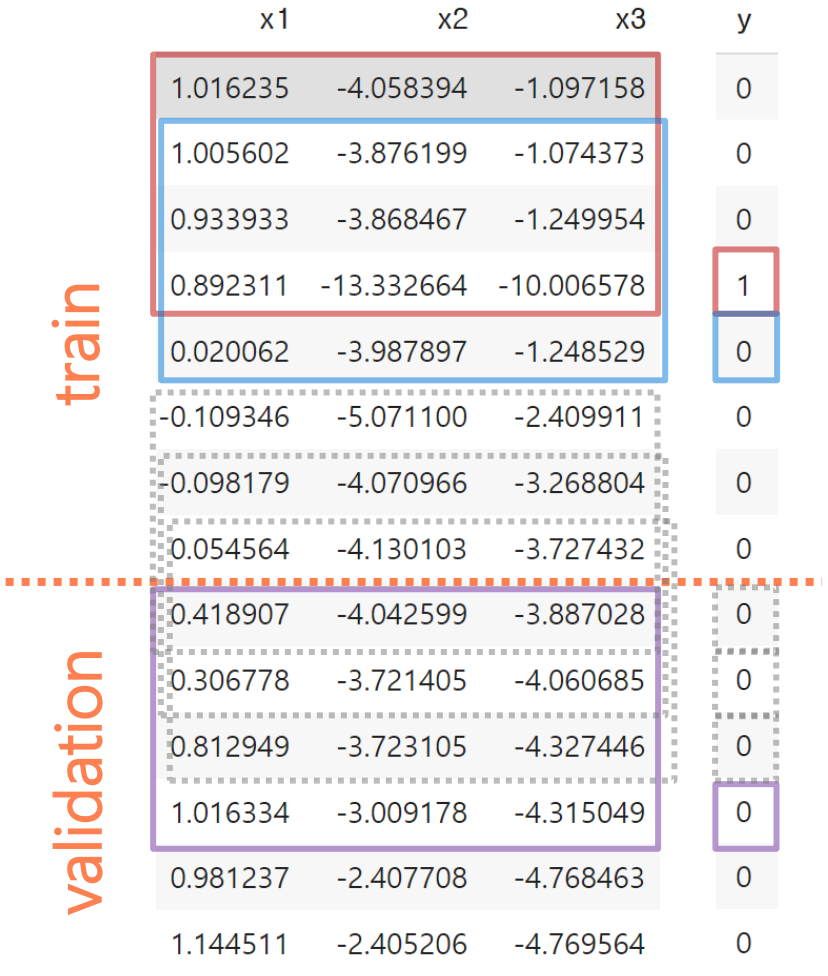
데이터 전처리② : 스케일링

✓ **3차원 X ①**

- train : val : test 분할 후
- 스케일링
- 3차원으로 변환

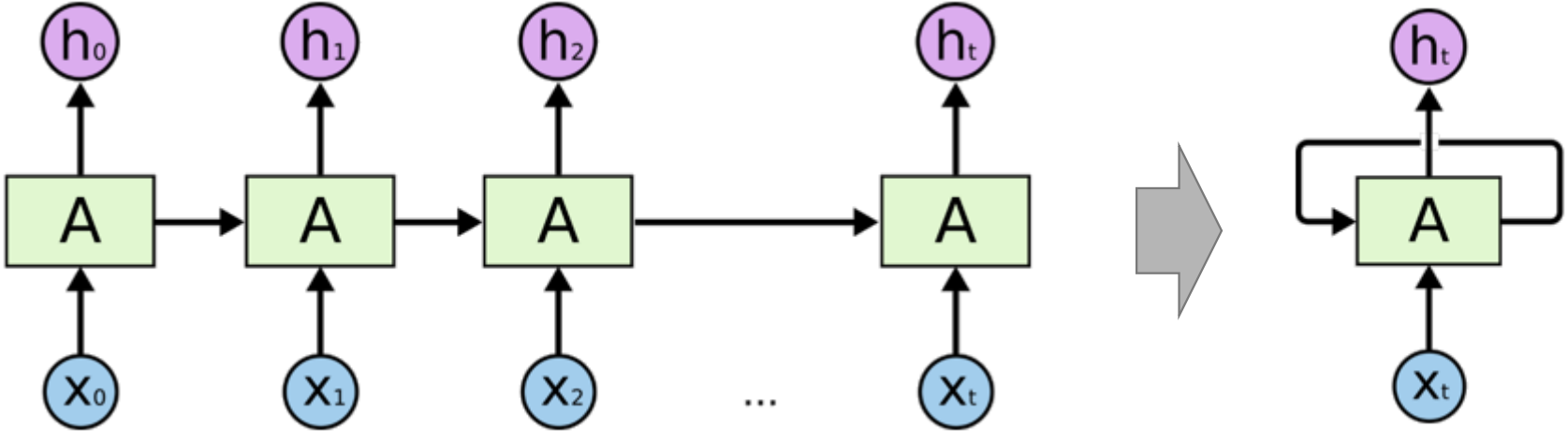
✓ **문제점**

- 데이터셋의 초기 데이터 손실



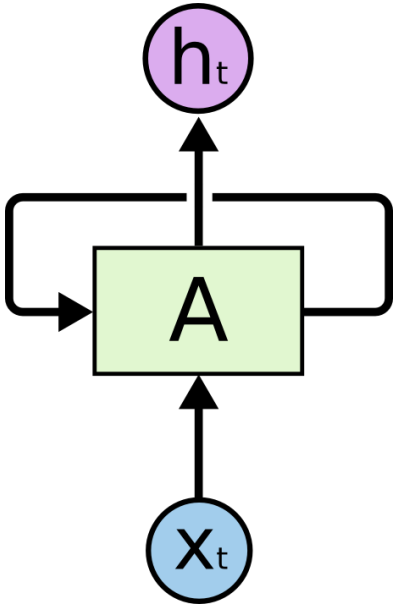
Recurrent Neural Networks

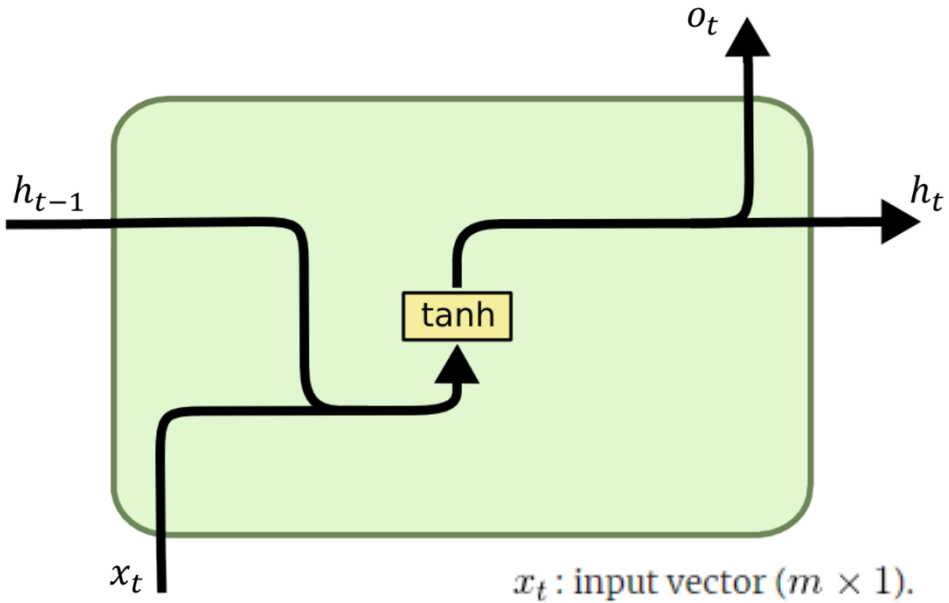
- ✓ **문제를 해결하기 위한, 의미 있는 기간 단위로 나눔**
  - 예, 과거 20일 단위(기간)의 정보를 이용하여 다음날의 주가를 예측한다.
    데이터셋에서 20개의 행을 의미, timesteps = 20
- ✓ **각 단계(Time Step)별로 학습하고 결과를 다음 단계로 전달**



Recurrent Neural Networks

✓과거의 정보를 현재에 반영해 학습하도록 설계





x_t : input vector $(m \times 1)$.
h_t : hidden layer vector $(n \times 1)$.
o_t : output vector $(n \times 1)$.

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Simple RNN

✓모델링 & 학습

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 10)                120
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================
Total params: 131
Trainable params: 131
Non-trainable params: 0
_________________________________________________________________
```

```
np.random.seed(2021)
tf.random.set_seed(2021)

# 세션클리어
keras.backend.clear_session()

# Sequential 모델 선언 + layer 추가하기(한꺼번에)
model = keras.models.Sequential([
    keras.layers.SimpleRNN(10, input_shape=[None, 1]),
    keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer=keras.optimizers.Adam(lr=0.005))
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_val, y_val))
```

Node 수(뉴런의 개수)

[time_step수, feature수]

Deep RNN

✓모델링 & 학습

	year	week	AvgTemp
0	2010	1	-3.000000
1	2010	2	-7.500000
2	2010	3	-7.900000
3	2010	4	-2.357143
4	2010	5	-3.342857
5	2010	6	-1.800000
6	2010	7	-0.314286
7	2010	8	-2.142857
8	2010	9	4.400000
9	2010	10	7.057143

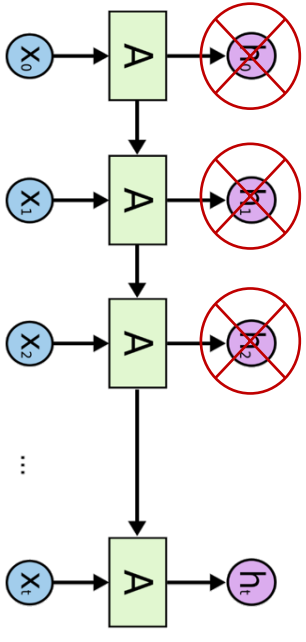
```
SimpleRNN(10, input_shape=[None, 1], return_sequences=False)
```

t=1 -3.000000

t=2 -7.500000

t=3 -7.900000

t=4 -2.357143



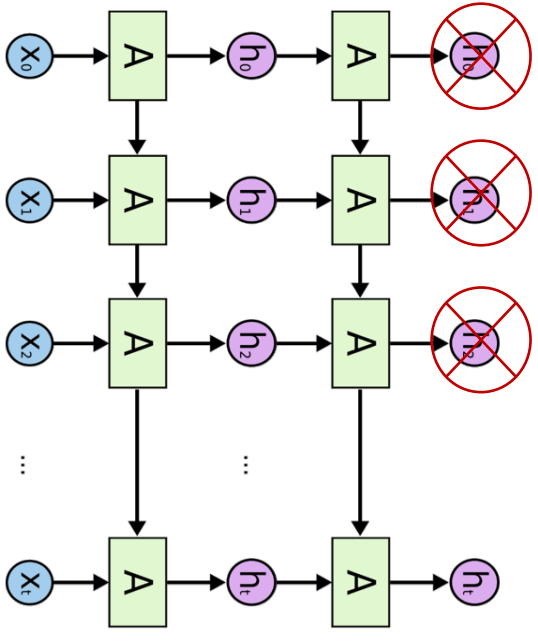
Deep RNN

✓모델링 & 학습

	year	week	AvgTemp
0	2010	1	-3.000000
1	2010	2	-7.500000
2	2010	3	-7.900000
3	2010	4	-2.357143
4	2010	5	-3.342857
5	2010	6	-1.800000
6	2010	7	-0.314286
7	2010	8	-2.142857
8	2010	9	4.400000
9	2010	10	7.057143

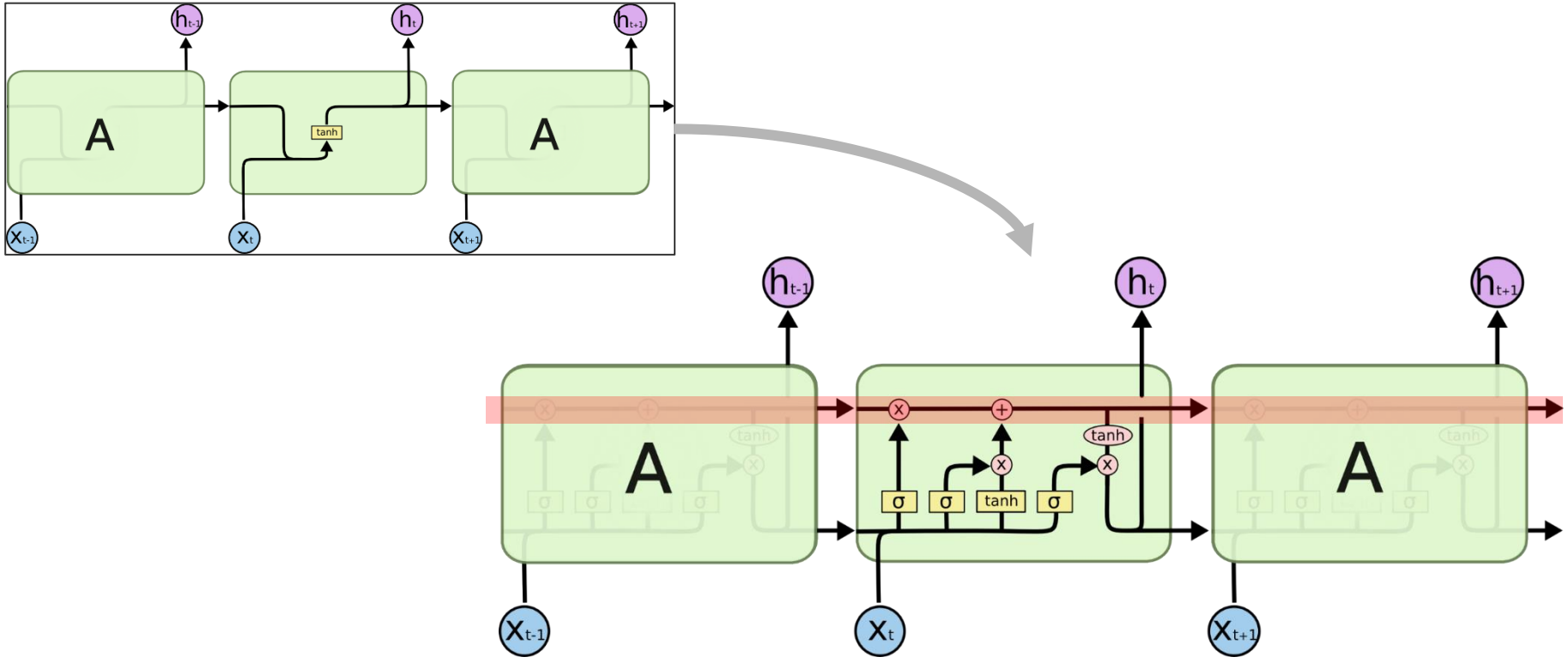
```
SimpleRNN(10, input_shape=[None, 1], return_sequences=True)
SimpleRNN(10, return_sequences=False),
```

t=1	-3.000000
t=2	-7.500000
t=3	-7.900000
t=4	-2.357143

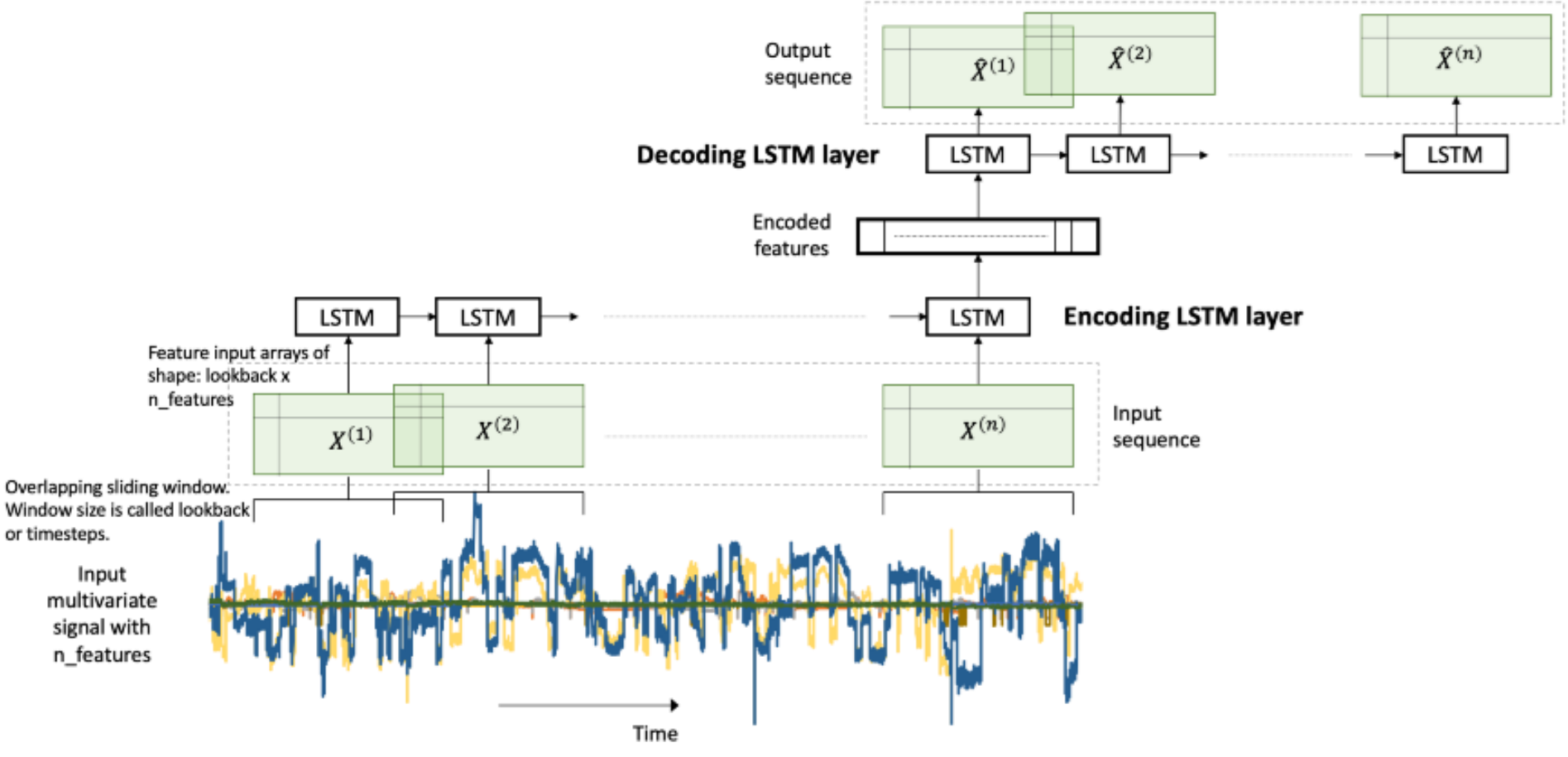


RNN의 Vanishing Gradient 문제

- ✓ 극복하기 위해서 고안된 것이 바로 LSTM
- ✓ RNN의 hidden state에 cell state(장기 기억 메모리)를 추가한 구조



LSTM + AE


Output sequence
$\hat{X}^{(1)}$
$\hat{X}^{(2)}$
$\hat{X}^{(n)}$
Decoding LSTM layer
LSTM
LSTM
LSTM
Encoded features
LSTM
LSTM
LSTM
Encoding LSTM layer
Feature input arrays of shape: lookback x n_features
$X^{(1)}$
$X^{(2)}$
$X^{(n)}$
Input sequence
Overlapping sliding window. Window size is called lookback or timesteps.
Input multivariate signal with n_features
Time

AE와 LSTM+AE 코드 구조 비교

AE

```
# Encoder
input_layer = Input(shape=(input_dim, ))
encoder = Dense(32, activation="relu")(input_layer)
encoder = Dense(16, activation="relu")(encoder)
# Decoder
decoder = Dense(32, activation="relu")(encoder)
decoder = Dense(input_dim)(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

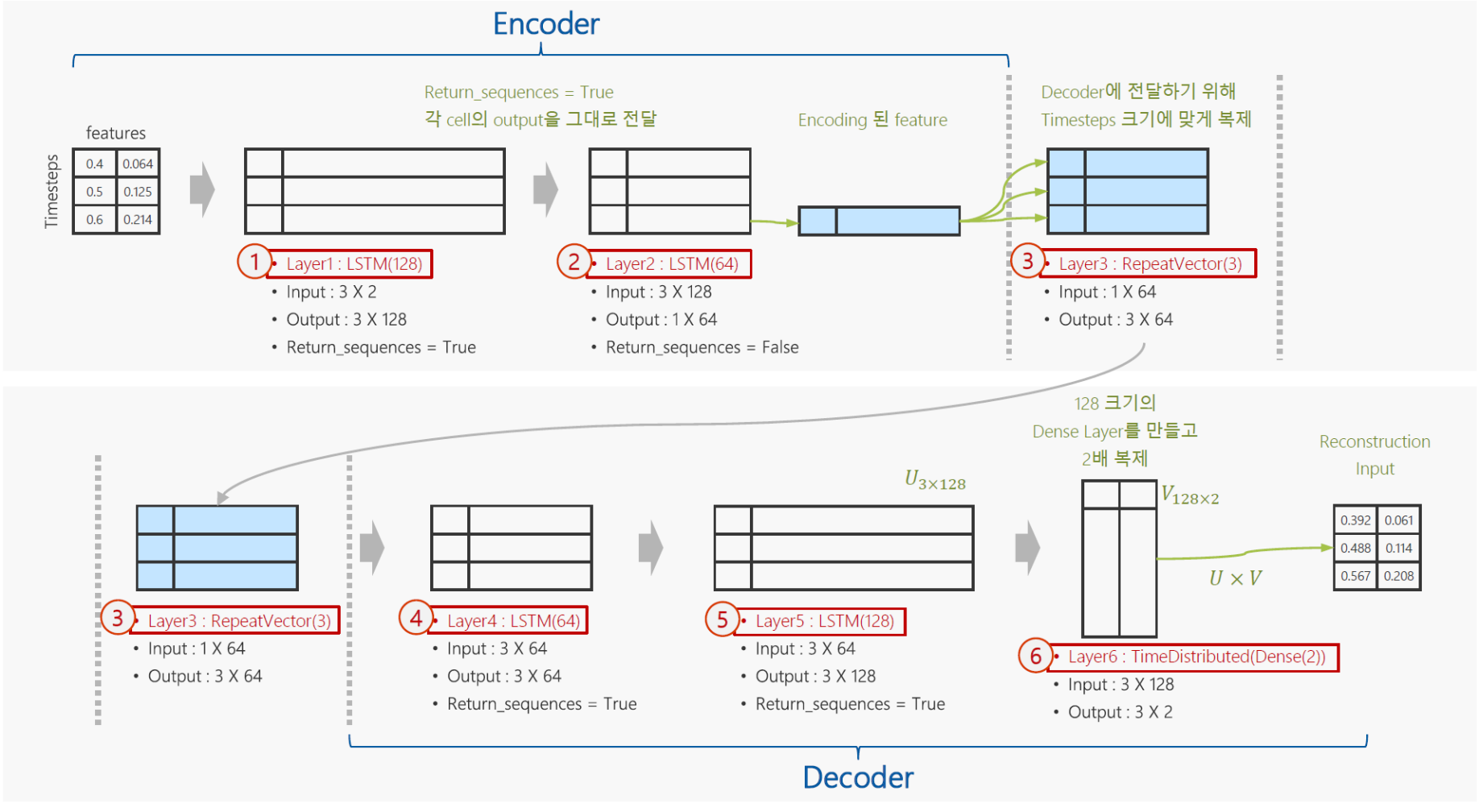
```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 59)]              0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1920
_________________________________________________________________
dense_5 (Dense)              (None, 16)                528
_________________________________________________________________
dense_6 (Dense)              (None, 32)                544
_________________________________________________________________
dense_7 (Dense)              (None, 59)                1947
=================================================================
```

LSTM + AE

```
# Encoder
input_layer = Input(shape=(timestep, n_features))
encoder = LSTM(32, return_sequences=True)(input_layer)
encoder = LSTM(16, return_sequences=False)(encoder)
encoder = RepeatVector(timestep)(encoder)
# Decoder
decoder = LSTM(16, return_sequences=True)(encoder)
decoder = LSTM(32, return_sequences=True)(decoder)
decoder = TimeDistributed(Dense(n_features))(decoder)
lstm_ae = Model(inputs=input_layer, outputs=decoder)
lstm_ae.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 10, 59)]          0
_________________________________________________________________
lstm_6 (LSTM)                (None, 10, 32)            11776
_________________________________________________________________
lstm_7 (LSTM)                (None, 16)                3136
_________________________________________________________________
repeat_vector_1 (RepeatVect  (None, 10, 16)            0
or)
_________________________________________________________________
lstm_8 (LSTM)                (None, 10, 16)            2112
_________________________________________________________________
lstm_9 (LSTM)                (None, 10, 32)            6272
_________________________________________________________________
time_distributed_1 (TimeDis  (None, 10, 59)            1947
tributed)
=================================================================
```

LSTM + AE 구조

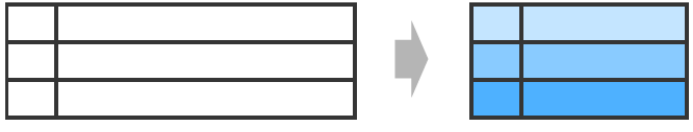

Encoder
features
Timesteps
0.4
0.064
0.5
0.125
0.6
0.214
Return_sequences = True
각 cell의 output을 그대로 전달
Encoding 된 feature
Decoder에 전달하기 위해
Timesteps 크기에 맞게 복제
1 Layer1 : LSTM(128)
Input : 3 X 2
Output : 3 X 128
Return_sequences = True
2 Layer2 : LSTM(64)
Input : 3 X 128
Output : 1 X 64
Return_sequences = False
3 Layer3 : RepeatVector(3)
Input : 1 X 64
Output : 3 X 64
128 크기의
Dense Layer를 만들고
2배 복제
$U_{3\times128}$
$V_{128\times2}$
$U \times V$
Reconstruction
Input
0.392
0.061
0.488
0.114
0.567
0.208
3 Layer3 : RepeatVector(3)
Input : 1 X 64
Output : 3 X 64
4 Layer4 : LSTM(64)
Input : 3 X 64
Output : 3 X 64
Return_sequences = True
5 Layer5 : LSTM(128)
Input : 3 X 64
Output : 3 X 128
Return_sequences = True
6 Layer6 : TimeDistributed(Dense(2))
Input : 3 X 128
Output : 3 X 2
Decoder

return_sequences

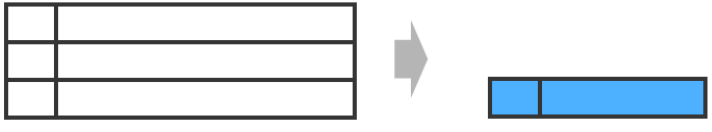
✓ return_sequences

- True : 입력 데이터의 차원 유지, 각 timestep 마다 생성된 hidden state 값을 사용.
- False : 입력 데이터의 차원이 1차원으로 줄어 듬.
  가장 마지막 timestep 에 의해 만들어진 hidden state 값을 사용.

```
# Encoder
input_layer = Input(shape=(timestep, n_features))
encoder = LSTM(128, return_sequences=True)(input_layer)
encoder = LSTM(64, return_sequences=False)(encoder)
encoder = RepeatVector(timestep)(encoder)
```



Return_sequences = True



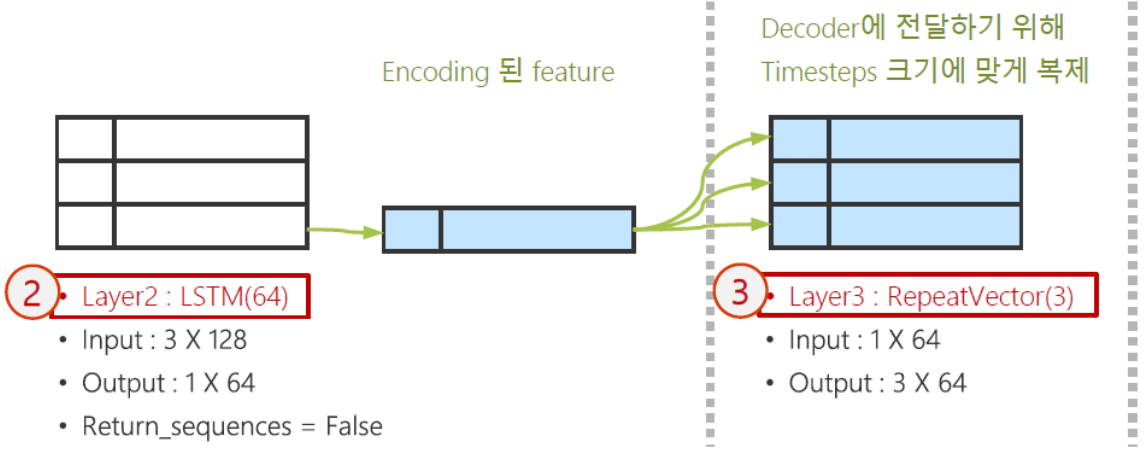
Return_sequences = False

RepeatVector

✓ RepeatVector(*timestep*)

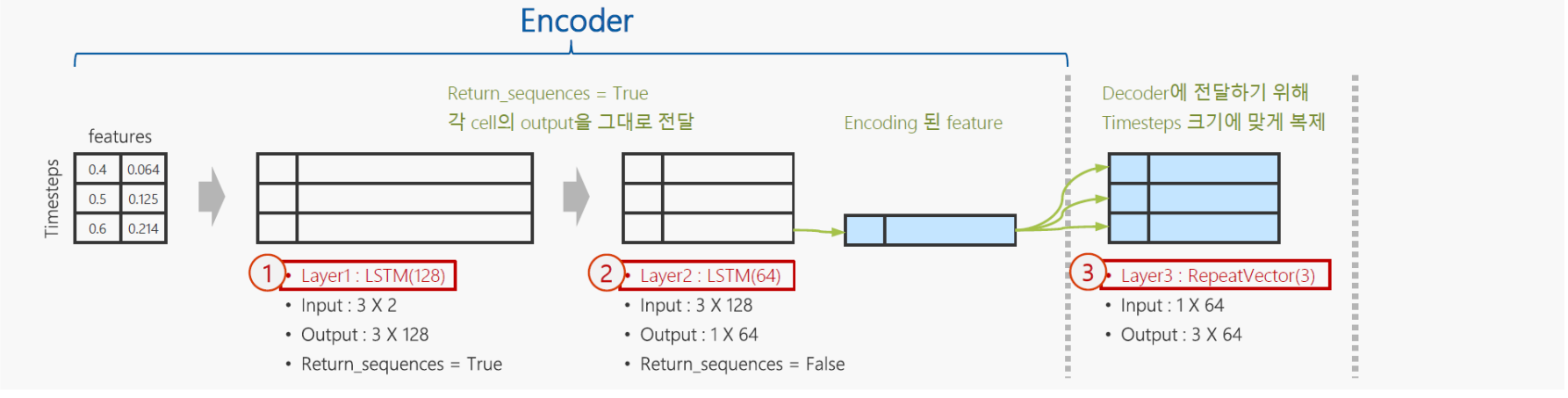
- return_sequences = False로 1차원이 된 데이터를
- 다음 LSTM으로 넘기기 위해 timestep 수 만큼 2차원 형태로 Replication.
- Encoder와 Decoder를 연결하는 bridge 역할

```
# Encoder
input_layer = Input(shape=(timestep, n_features))
encoder = LSTM(128, return_sequences=True)(input_layer)
encoder = LSTM(64, return_sequences=False)(encoder)
encoder = RepeatVector(timestep)(encoder)
```



Encoder

```
# Encoder
input_layer = Input(shape=(timestep, n_features))
① encoder = LSTM(128, return_sequences=True)(input_layer)
② encoder = LSTM(64, return_sequences=False)(encoder)
③ encoder = RepeatVector(timestep)(encoder)
```

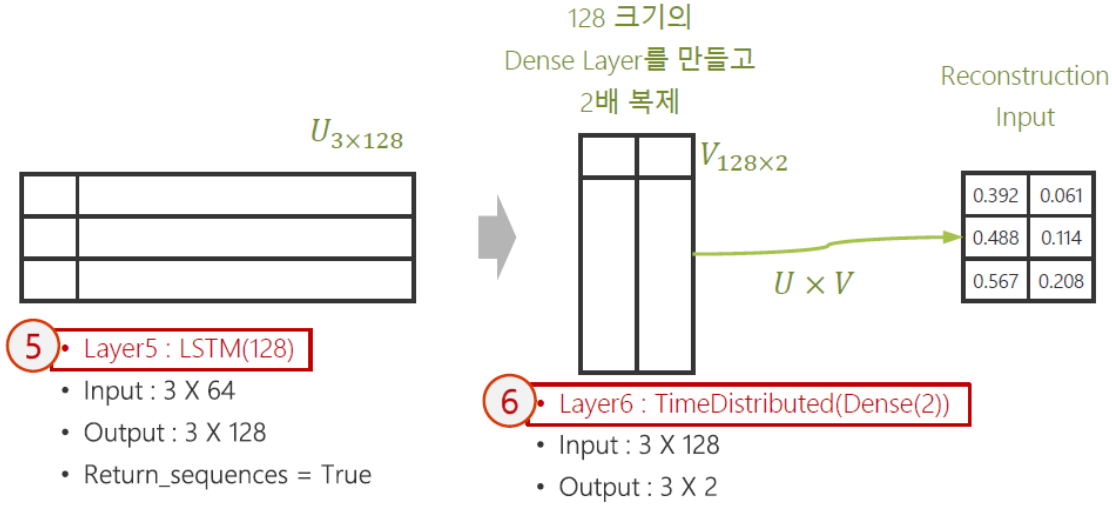


②의 Output 을 압축된 특징벡터라고 부른다. ➔ 다른 지도학습이나 비지도 학습을 위해 사용될 수 있음.

TimeDistributed

- 이전 레이어에서 출력된 피처 수와 동일한 길이의 벡터를 생성하고,
- Input 데이터의 feature 수 만큼 복제하여 행렬을 생성.
- Input의 shape 와 동일하게 출력

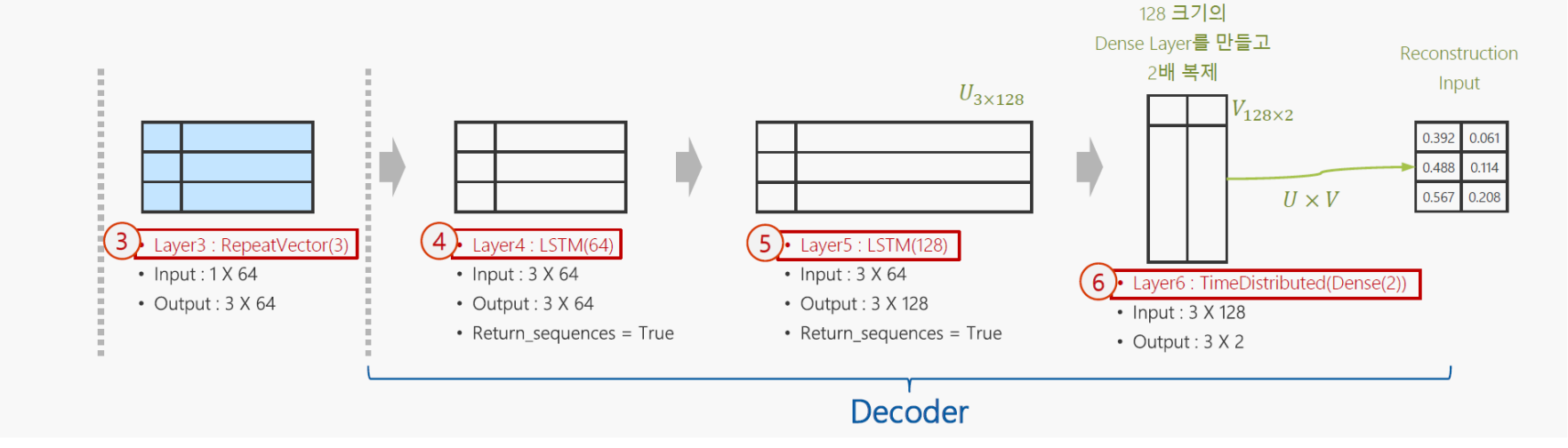
```
# Decoder
decoder = LSTM(64, return_sequences=True)(encoder)
decoder = LSTM(128, return_sequences=True)(decoder)
decoder = TimeDistributed(Dense(n_features))(decoder)
```



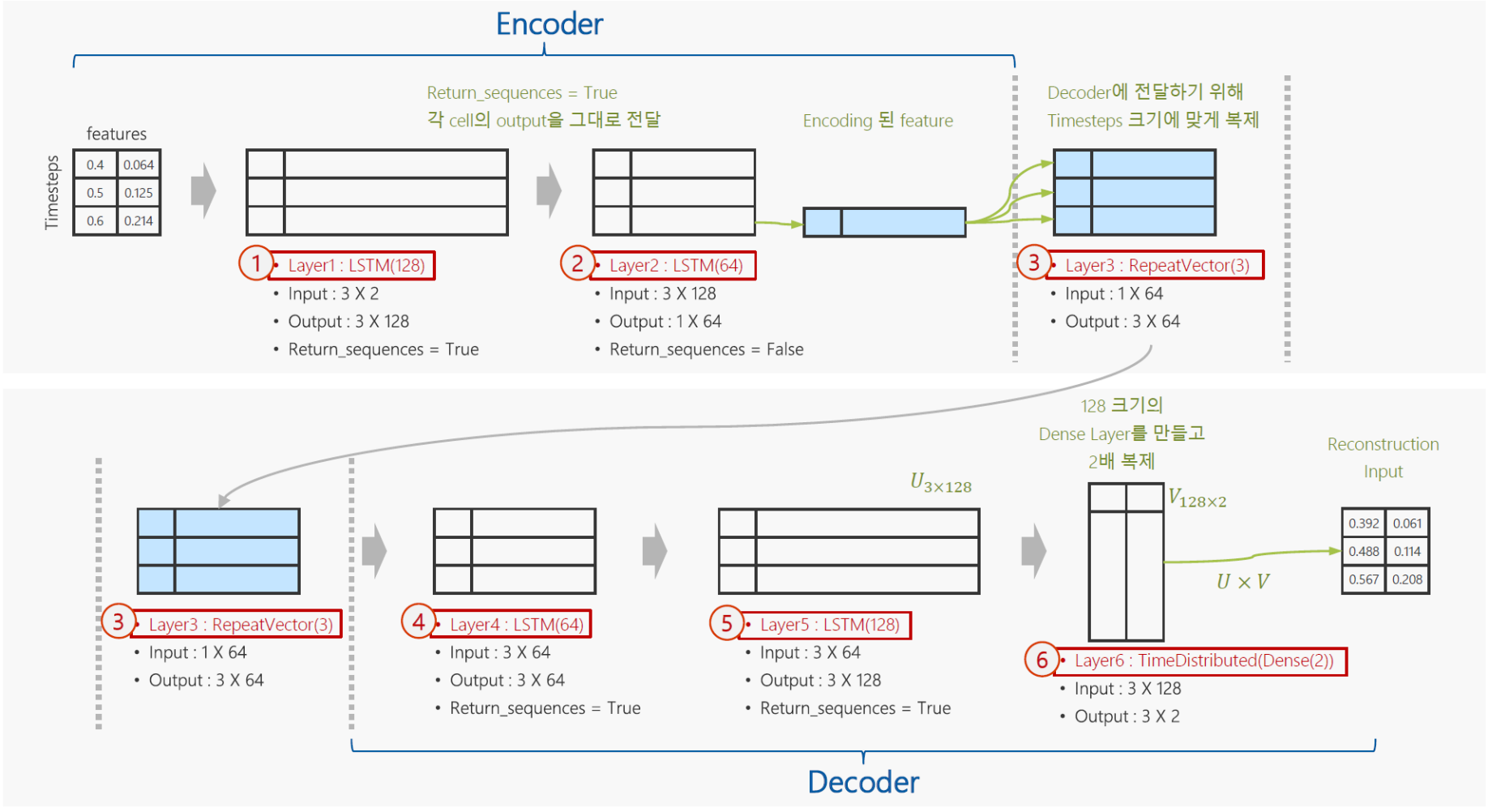
Decoder

- Decoder는 encoder 구조의 역(inverse) 입니다.
- 마지막은 Input 과 동일한 구조로 Output 을 만들어 냅니다.

```
# Decoder
4 decoder = LSTM(64, return_sequences=True)(encoder)
5 decoder = LSTM(128, return_sequences=True)(decoder)
6 decoder = TimeDistributed(Dense(n_features))(decoder)
```



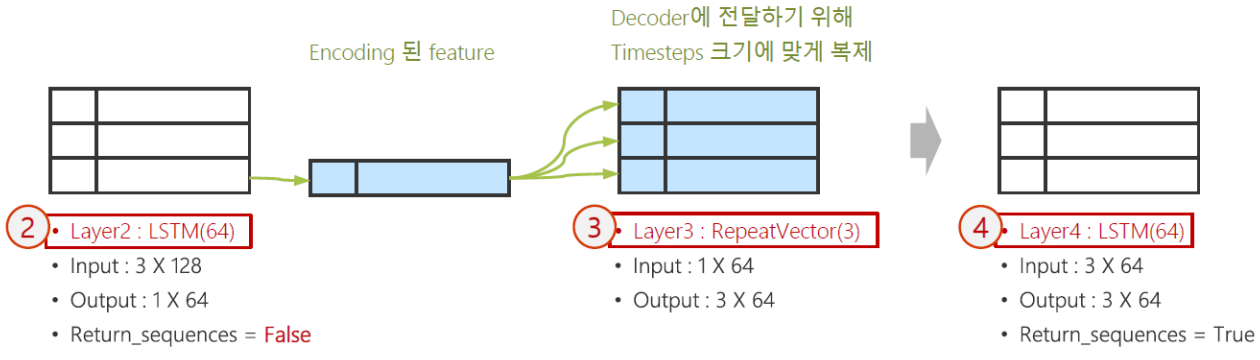
LSTM + AE 구조 - Review



LSTM + AE 구조의 변형

방법1

압축된 Layer에서
마지막 Output만 사용해서
Decoding



방법2

압축된 Layer에서
각 Time step 별 Output을
모두 사용해서 Decoding