CHAPTER

15 RNN과 CNN을 사용해 시퀀스 처리하기 (상)

둘러보기

1. 순환 뉴런과 순환 층
2. RNN 훈련하기
3. 시계열 예측하기
4. 긴 시퀀스 다루기 (하)

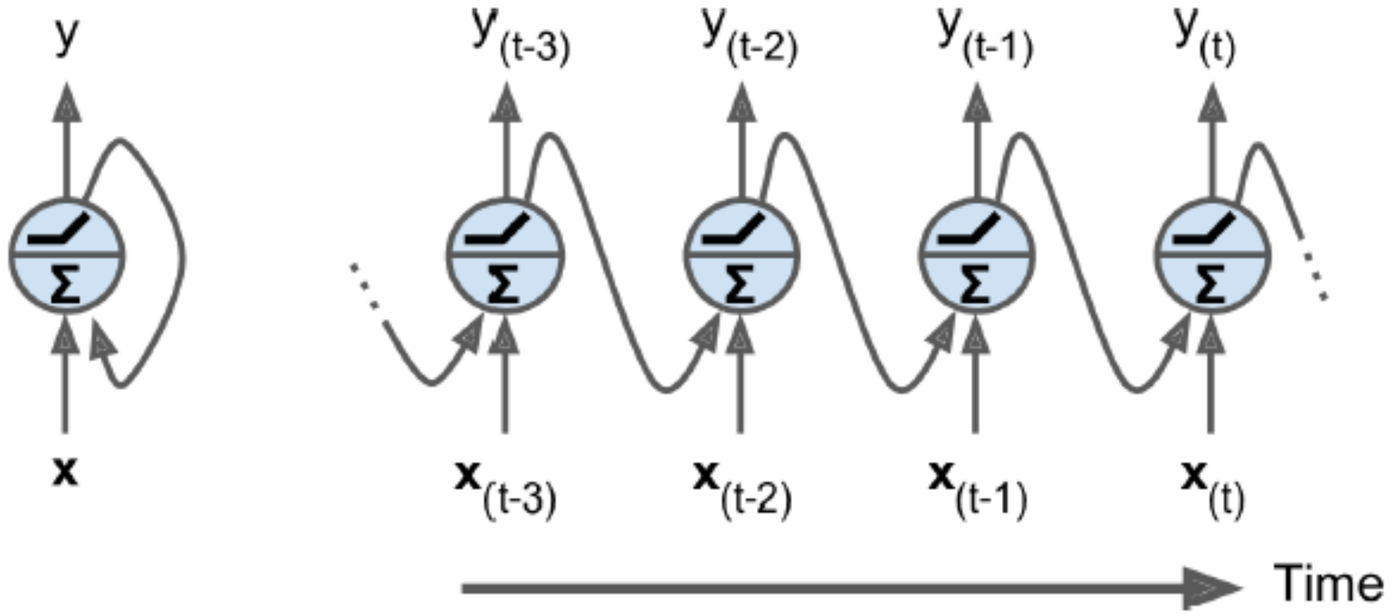
0. 기초 용어

순환 신경망(RNN, recurrent neural networks)

미래를 어느 정도 예측할 수 있는 네트워크 솔루션. 일반적으로 이 신경망은 임의 길이를 가진 시퀀스를 다룰 수 있다. 때문에 자연어 처리(NLP)에 매우 유용하다.

1. 순환 뉴런과 순환 층

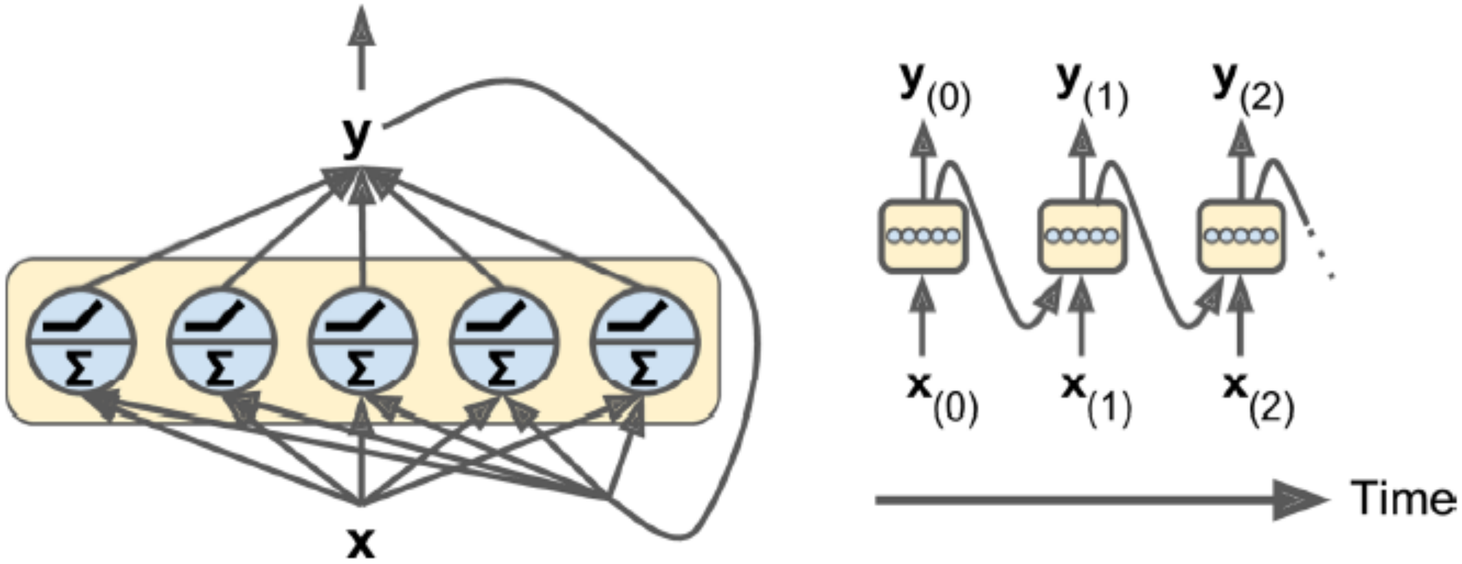
지금까지는 활성화 신호가 입력층에서 출력층 한 방향으로만 흐르는 피드포워드 신경망 위주였는데, 순환 신경망은 뒤쪽으로 순환하는 연결도 있다는 차이점이 있음.



▲ 순환 뉴런(왼쪽)과 타임 스텝으로 펼친 모습(오른쪽)
오른쪽 : "시간에 따라 네트워크를 펼쳤다"

1. 순환 뉴런과 순환 층

타임 스텝 t마다 모든 뉴런은 입력 벡터와 이전 타임 스텝의 출력 벡터를 받는다. 이러면 입력과 출력이 모두 벡터가 된다. (뉴런이 하나일 때는 출력이 스칼라이다)



▲ 순환 뉴런으로 된 층(왼쪽)과 타임 스텝으로 펼친 모습(오른쪽)

1. 순환 뉴런과 순환 층

타임 스텝 t마다 모든 뉴런은 입력 벡터와 이전 타임 스텝의 출력 벡터를 받는다. 이러면 입력과 출력이 모두 벡터가 된다. (뉴런이 하나일 때는 출력이 스칼라이다)

$$\mathbf{y}_{(t)} = \phi\left({\mathbf{x}_{(t)}}^{T} \cdot \mathbf{w}_x + {\mathbf{y}_{(t-1)}}^{T} \cdot \mathbf{w}_y + b\right)$$

▲ 순환 층 전체의 출력 벡터를 계산하는 식

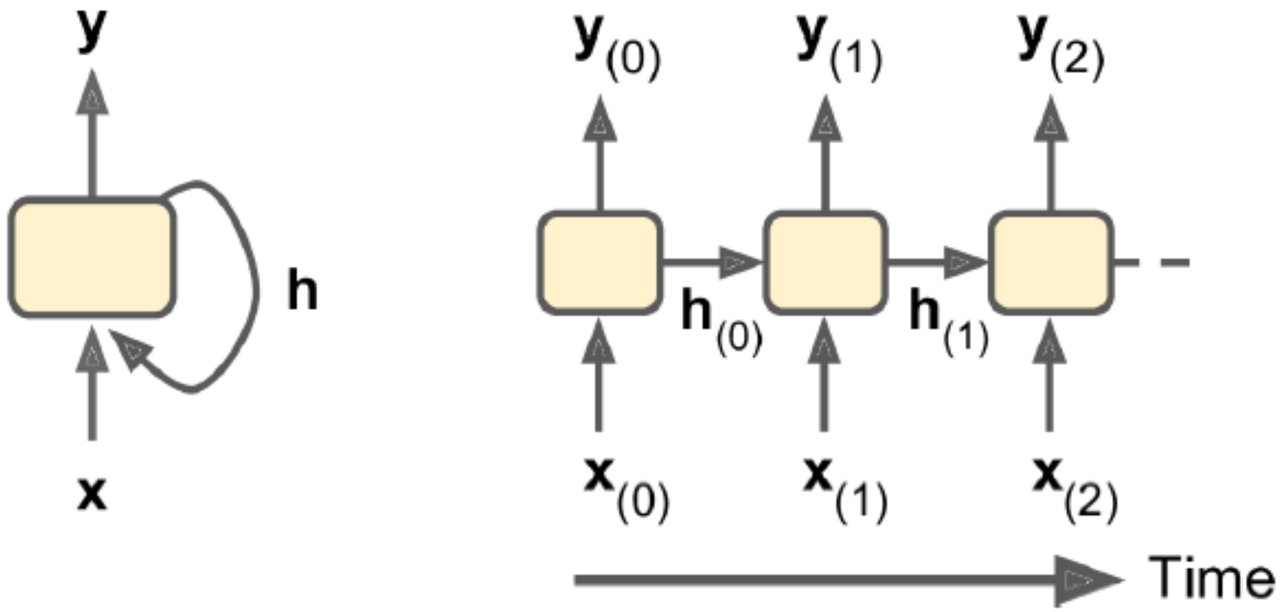
$$\begin{aligned}\mathbf{Y}_{(t)} &= \phi\left(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b}\right) \\ &= \phi\left(\left[\mathbf{X}_{(t)} \quad \mathbf{Y}_{(t-1)}\right] \cdot \mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x \\ \mathbf{W}_y\end{bmatrix}\end{aligned}$$

▲ 미니배치에 있는 전체 샘플에 대한 순환 뉴런 층의 출력을 계산하는 식

1. 순환 뉴런과 순환 층

메모리 셀

타임 스텝에 걸쳐서 어떤 상태를 보존하는 신경망의 구성 요소. 하나의 순환 뉴런 또는 순환 뉴런의 층은 짧은 패턴만 학습할 수 있다.



1. 순환 뉴런과 순환 층

입력과 출력 시퀀스

시퀀스-투-시퀀스 네트워크 : 입력 시퀀스를 받아 출력 시퀀스를 만들 수 있다.

시퀀스-투-벡터 네트워크 : 입력 시퀀스를 네트워크에 주입, 마지막을 제외한 모든 출력을 무시

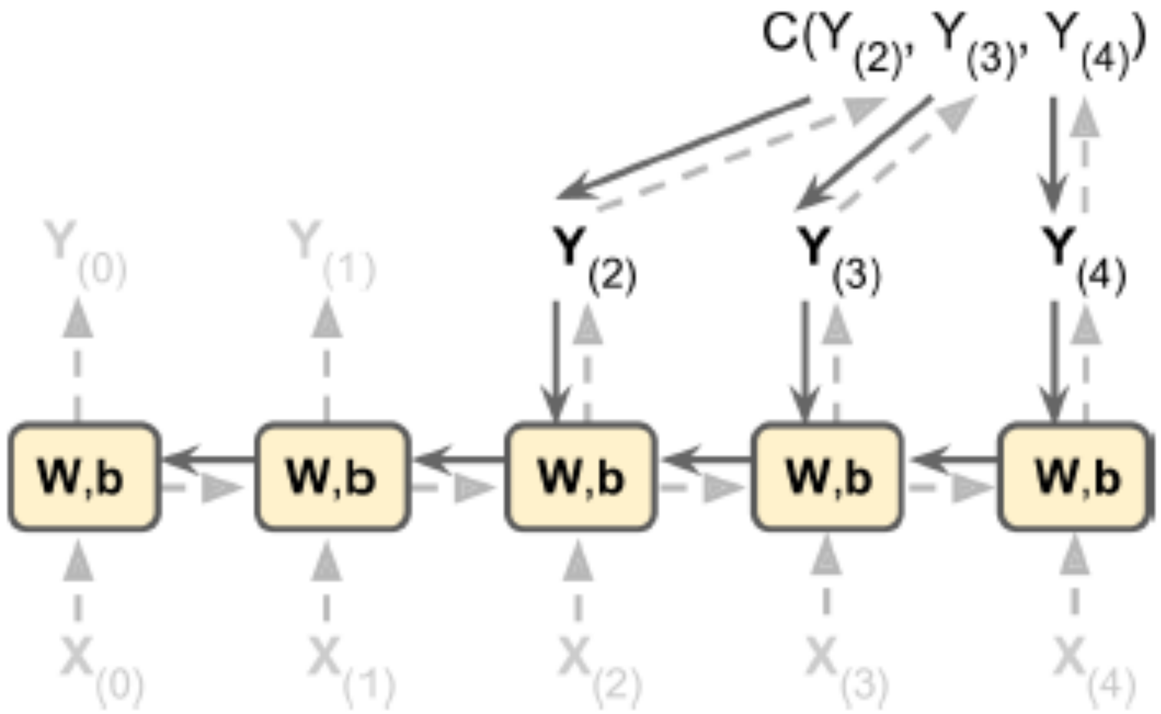
벡터-투-시퀀스 네트워크 : 각 타임 스텝에서 하나의 입력 벡터를 반복해서 네트워크에 주입하고, 하나의 시퀀스를 출력할 수 있다.

인코더-디코더 : 인코더라 부르는 시퀀스-투-벡터 네트워크 뒤에 디코더라 부르는 벡터-투-시퀀스 네트워크를 연결할 수 있다.

2. RNN 훈련하기

BPTT (backpropagation through time)

RNN을 훈련하기 위한 기법으로 타임 스텝으로 네트워크를 펼치고 보통의 역전파를 사용하는 것.



3. 시계열 예측하기

시계열 (time series)

데이터의 타임 스텝마다 하나 이상의 값을 가진 시퀀스.

단변량 시계열 (univariate time seires)

타임 스텝마다 하나의 값을 가진 시퀀스.

다변량 시계열 (multivariate time seires)

타임 스텝마다 여러 개의 값을 가진 시퀀스.

값 대체 (imputation)

과거 데이터에서 누락된 값을 예측.

3. 시계열 예측하기

시계열 생성

요청한 만큼 n_steps 길이의 여러 시계열을 만드는 함수.

```
def generate_time_series(batch_size, n_steps):
    freq1, freq2, offsets1, offsets2 = np.random.rand(4, batch_size, 1)
    time = np.linspace(0, 1, n_steps)
    series = 0.5 * np.sin((time - offsets1) * (freq1 * 10 + 10))  #   wave 1
    series += 0.2 * np.sin((time - offsets2) * (freq2 * 20 + 20)) # + wave 2
    series += 0.1 * (np.random.rand(batch_size, n_steps) - 0.5)   # + noise
    return series[..., np.newaxis].astype(np.float32)
```

3. 시계열 예측하기

훈련 세트, 검증 세트, 테스트 세트 만들기

X_train은 7,000개의 시계열을 담고, X_valid는 2,000개, X_test는 1,000개를 담는다.

```
n_steps = 50
series = generate_time_series(10000, n_steps + 1)
X_train, y_train = series[:7000, :n_steps], series[:7000, -1]
X_valid, y_valid = series[7000:9000, :n_steps], series[7000:9000, -1]
X_test, y_test = series[9000:, :n_steps], series[9000:, -1]
```

3. 시계열 예측하기

기준 성능

RNN을 시작하기 전에 기준 성능을 몇 개 준비하는 것이 좋다. 가장 간단한 방법은 각 시계열의 마지막 값을 그대로 예측하는 **순진한 예측**방법이다. 또 다른 방법으로는 **완전 연결 네트워크**이다.

```
>>> y_pred = X_valid[:, -1]
>>> np.mean(keras.losses.mean_squared_error(y_valid, y_pred))
0.020211367
```

▲ 순진한 예측. 이 예측의 경우 평균 제곱 오차는 0.020 정도이다.

```
>>> model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[50, 1]), keras.layers.Dense(1)])
0.00414548622442721
```

▲ 완전 연결 네트워크. 이 네트워크는 입력마다 1차원 특성 배열을 기대하기 때문에 flatten 층을 추가해야 한다. 순진한 예측보다 더 나은 결과를 보인다.

3. 시계열 예측하기

간단한 RNN 구현하기

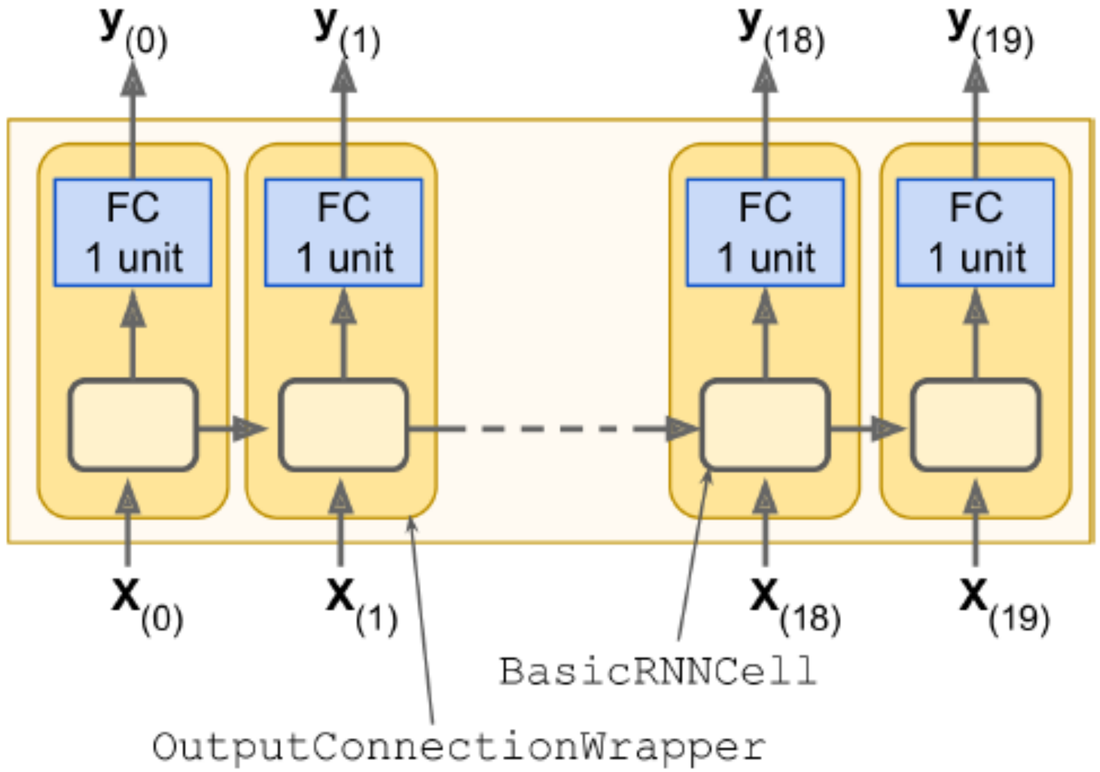
이 모델을 훈련, 평가하면 0.014에 달하는 평균 제곱 오차값을 얻는다. 이는 순진한 예측보다는 낫지만, 간단한 선형 모델을 앞지르지는 못한다.

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(1, input_shape=[None, 1])
])
```

▲ 가장 간단한 RNN 을 구현한 코드.

3. 시계열 예측하기

심층 RNN



▲ 심층 RNN을 타임 스텝으로 펼친 모습

3. 시계열 예측하기

심층 RNN 모델을 컴파일, 훈련, 평가하면 0.003의 평균 제곱 오차값을 얻는다. 이를 통해 심층 RNN이 선형 모델을 확실히 앞지르는 성능임을 확인할 수 있다.

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
```

▲ tf.keras로 심층 RNN을 구현하는 코드.

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])
```

▲ 성능을 더 개선한 코드. return_sequences = True를 제거하고 마지막 코드에서 keras.layers.Dense(1) 로 바꾼다.

3. 시계열 예측하기

여러 타임 스텝 앞을 예측하기

지금까지는 다음 타임 스텝의 값만 예측했지만 타깃을 적절히 바꾸면 여러 타임 스텝 앞의 값을 예측할 수 있다. 여러 타임 스텝 앞의 값을 예측하는 방법으로는 두 가지가 있다.

1. 이미 훈련된 모델을 사용하여 다음 값을 예측한 다음 이 값을 입력으로 추가하는 것.
2. RNN을 훈련해 다음 값 10개를 한 번에 예측하는 것.

3. 시계열 예측하기

방법 1 : 이미 훈련된 모델을 사용

다음 스텝에 대한 예측은 보통 더 미래의 타임 스텝에 대한 예측보다 정확하다. 미래의 타임 스텝은 오차가 누적될 수 있기 때문이다. 이 모델은 한 번에 하나의 미래 스텝을 예측하기 위해 RNN을 사용하는 것보다 낫다.

```
series = generate_time_series(1, n_steps + 10)
X_new, Y_new = series[:, :n_steps], series[:, n_steps:]
X = X_new
for step_ahead in range(10):
    y_pred_one = model.predict(X[:, step_ahead:])[:, np.newaxis, :]
    X = np.concatenate([X, y_pred_one], axis=1)

Y_pred = X[:, n_steps:]
```

3. 시계열 예측하기

방법 2 : RNN을 훈련하여 다음 값 10개를 한 번에 예측

시퀀스-투-벡터 모델을 사용하지만 1개가 아니라 10개를 출력해야 한다.

1. 타깃을 다음 10개의 값이 담긴 벡터로 바꾼다.
2. 10개의 유닛을 가진 출력층을 만든다.
3. 이 모델을 훈련하면 한 번에 다음 값 10개를 예측할 수 있다.

```
series = generate_time_series(10000, n_steps + 10)
X_train, Y_train = series[:7000, :n_steps], series[:7000, -10:, 0]
X_valid, Y_valid = series[7000:9000, :n_steps], series[7000:9000, -10:, 0]
X_test, Y_test = series[9000:, :n_steps], series[9000:, -10:, 0]
```

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(10)
])
```

3. 시계열 예측하기

방법 2 개선 : 마지막 타임 스텝만이 아닌 모든 타임 스텝에서 예측하기

시퀀스-투-시퀀스 RNN 활용하여 모든 타임 스텝에서 다음 값 10개를 예측하도록 모델을 만들면 모든 타임 스텝에서 RNN 출력에 대한 항이 손실에 포함되므로 훈련을 안정적으로 만들고 훈련 속도를 높일 수 있다.

```
Y = np.empty((10000, n_steps, 10))
for step_ahead in range(1, 10 + 1):
    Y[..., step_ahead - 1] = series[..., step_ahead:step_ahead + n_steps, 0]
Y_train = Y[:7000]
Y_valid = Y[7000:9000]
Y_test = Y[9000:]
```

▲ 타깃 시퀀스. 각 타깃은 입력 시퀀스와 동일한 길이의 시퀀스이다.

▼ 개선된 모델.

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

3. 시계열 예측하기

MSE만을 계산하는 사용자 정의 지표 사용

훈련하는 동안 모든 출력이 필요하지만, 예측 및 평가에는 마지막 타임 스텝의 출력만 사용된다.

평가를 위해 마지막 타임 스텝의 MSE만을 계산하는 사용자 정의 지표를 사용한다. 검증 평균 제곱 오차로 0.006이 나오며 이는 이전 모델보다 25% 향상된 값이다.

```
def last_time_step_mse(Y_true, Y_pred):
    return keras.metrics.mean_squared_error(Y_true[:, -1], Y_pred[:, -1])

optimizer=keras.optimizers.Adam(lr=0.01)
model.compile(loss="mse", metrics=[last_time_step_mse])
```