# QUANTITATIVE ISSUE

2026. 1. 21

MACRO팀

김동영, CFA Quant Analyst
dy76.kim@samsung.com

리서치센터 리포트
바로가기

# 퀀트 모델링 A to Z

## (8) RNN(순환 신경망) AI 투자 모델

- 시계열 혹은 순차 데이터의 모델링이 뛰어난 "RNN" 계열 모델의 설명과 활용법 소개

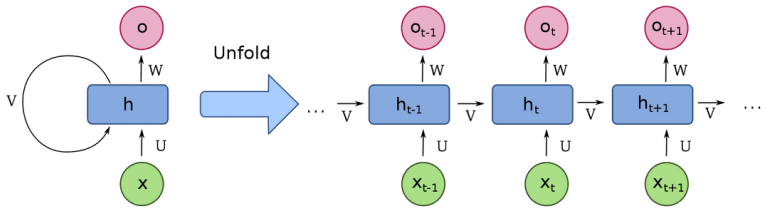
## WHAT'S THE STORY?

RNN(Recurrent Neural Network, 순환 신경망)이란, 순차 데이터(순서가 있는 데이터)를 다루기 위해 고안된 특수한 신경망 모델을 말한다. 과거 계량 경제 분야에서는 시계열 분석을 위해 ARIMA, VAR(Vector Autoregression), ARCH, GARCH 등 다양한 모델을 사용한 바 있다. 딥러닝 측면에서는 RNN 계열이 이런 시계열 분석을 대체할 수 있다. 또한, RNN은 문장 생성과 같은 자연어 기준의 순차 데이터 모델링에도 강점을 가지고 있다.

이런 순환 신경망은 Simple RNN부터 시작하여 LSTM 등 다양한 모델이 나왔으며, 이것들이 계속 발전하여 어텐션, 트랜스포머를 지나, 현재의 ChatGPT 같은 대화형 인공지능의 성공에까지 연결된다.

본 자료는, 순환 신경망의 가장 기초인 Simple RNN 모델에 대해서 설명하고, 금융 시계열 데이터에 이를 활용하는 방법과 활용 시의 유의점 등에 대해 예시를 통해 상세 정리하고자 한다.

### RNN 모델 개념도




자료: https://theaisummer.com/understanding-lstm/

삼성증권

# I. RNN 모델 구조

본 리포트는, 순환/재귀 신경망의 가장 기초인 Simple RNN 모델에 대해서 설명하고, 금융 시계열 데이터에 이를 활용하는 방법과 활용 시의 유의점 등에 대해 예시를 통해 상세 정리하고자 한다.

RNN이 속한 "딥러닝 모델"들의 기초적인 이해 및 공통적인 사용법에 대해서는, 26/1/2일 자 "퀀트 모델링 A to Z - (6) MLP(Multi-Layer Perceptron) AI 투자 모델" 리포트를 먼저 확인하는 것을 추천한다.

여기서 사용 및 설명하는 전체 코드의 프로그래밍 환경은 다음과 같다.

```
Python 3.12.7
tensorflow 2.20.0
tensorflow.keras 3.12.0
```

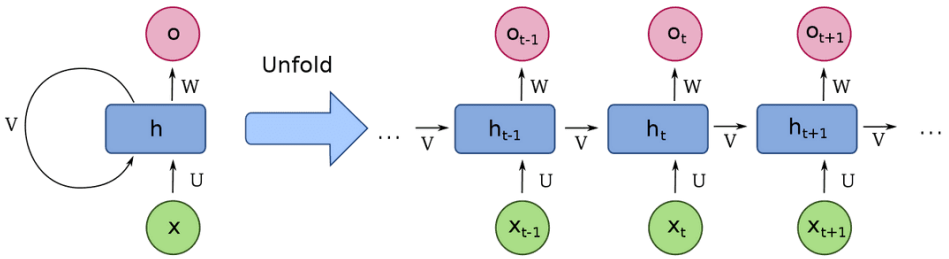
## 1) RNN의 이해

순차 데이터는, 시계열 데이터나 문장 텍스트와 같이 순서가 의미를 가지는 데이터를 말한다. 딥러닝 분야에서는, "이후 정보가 이전 정보의 영향을 받는 형태인" 이런 순차 데이터의 모델링을 위해 RNN(Recurrent Neural Network, 순환/재귀 신경망) 분야가 개발되었다.

과거 금융 시계열에 대해 VAR나 ARIMA 모델이 하던 일을, 딥러닝에서는 RNN 계열 모델이 비슷하게 혹은 더 뛰어나게 분석 및 예측 작업을 수행할 수 있다. 본 자료에서 논의하는 "좁은 의미의 RNN 모델"은, 순환 신경망들 중에서 가장 기초에 해당하는 Simple RNN(Vanilla RNN) 모델을 지칭하는 것이며, 이후부터는 RNN으로만 표기하겠다.

RNN(Simple RNN) 모델은, 신경망의 은닉층이 작동할 때 1) 데이터의 '이전 상태'를 내부에 저장하고, 2) 노드 계산 시 '이전 상태'를 입력으로 같이 받아서 계산하여, '이전 상태'가 일정 부분 반영되도록 만들어진 신경망이다.

**RNN 모델 개념도**




자료: https://theaisummer.com/understanding-lstm/

RNN 모델의 경우에는, 수식을 통해 개념을 이해하는 것이 훨씬 편리하다.

## A. 단순 MLP의 수식 표현

RNN 설명을 하기 이전에 복습을 위해, 가장 기본적인 신경망인 "MLP"에서의 한 개 층의 계산을 수식으로 표현하면 다음과 같다.

**MLP 계산의 수식**

**[ 가정 ]**

입력 데이터가 d 차원의 벡터임. $\boldsymbol{x} = [x_1\ x_2\ \dots\ x_d]$

은닉층 출력 데이터가 m 차원의 벡터임. $\boldsymbol{h} = [h_1\ h_2\ \dots\ h_m]$

은닉층 출력 데이터와 입력 데이터 간의 관계식은 다음과 같음.

**[ 행렬 표현 ]**

$$\boldsymbol{h} = f(\boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b})$$

이때, $\boldsymbol{W}$는 가중치 행렬.

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{d1} & \cdots & w_{dm} \end{bmatrix}$$

$\boldsymbol{b}$는 편향 벡터.

$$\boldsymbol{b} = [b_1\ b_2\ \dots\ b_m]$$

$f$는 활성화 함수(tanh 함수, 시그모이드 함수 같은 미리 정한 임의의 비선형 함수).

**[ 원소 단위 표현]**

$$[h_1\ h_2\ \dots\ h_m] = f\left([x_1\ x_2\ \dots\ x_d] \cdot \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{d1} & \cdots & w_{dm} \end{bmatrix} + [b_1\ b_2\ \dots\ b_m]\right)$$

**설명:**

d 차원의 입력 벡터와 m 차원의 은닉층 출력 벡터는 위의 수식으로 연결됨.

$\boldsymbol{x}$ 데이터가 들어오면, 위 계산식을 통해서 $\boldsymbol{h}$ 가 출력됨($\boldsymbol{x}$에 가중치 행렬을 곱하고 편향을 더한 다음에, 나온 값에 정해진 $f$ 함수 계산만 하면 $\boldsymbol{h}$ 가 나옴. 전체 연산은 단순히 행렬 연산의 반복에 불과함).

신경망 학습은, 역전파 과정을 통해서, 최적의 $\boldsymbol{W}$($d \times m$ 행렬)와 $\boldsymbol{b}$를 찾아가는 과정임.
즉, 전체 학습 파라미터 수는 $\boldsymbol{W}$의 $d \times m$개와 $\boldsymbol{b}$의 $m$개, 즉 $d \times m + m$개임.

배치 입력(N개의 샘플이 한 번에 들어오는 형태)을 가정한다면, $\boldsymbol{h}$ 벡터 대신 N만큼 세로로 늘어나는 $\boldsymbol{H}$ 형렬, $\boldsymbol{x}$ 벡터 대신 N만큼 세로로 늘어나는 $\boldsymbol{X}$ 형렬, $\boldsymbol{b}$ 벡터 대신 N만큼 세로로 늘어나는 $\boldsymbol{1b}$ 형렬로 수식을 바꾸면 됨.

## B. RNN의 수식 표현

RNN의 은닉층 계산에서는, $\boldsymbol{x}$만 입력으로 받는 것이 아니라 은닉층 자체의 직전 timestep(직전 시점) 데이터도 입력으로 같이 받아서 계산한다. 이를 수식으로 쓰면 다음과 같다.

**RNN 계산의 수식**

**[ 가정 ]**

입력 데이터가 d 차원의 벡터임. $\boldsymbol{x} = [x_1\ x_2\ \dots\ x_d]$.

이때 $t$ 시점별로 데이터가 존재함. $\boldsymbol{x_t} = [x_{1t}\ x_{2t}\ \dots\ x_{dt}]$

은닉층 출력 데이터가 m 차원의 벡터임. $\boldsymbol{h} = [h_1\ h_2\ \dots\ h_m]$

이때 $t$ 시점별로 데이터가 존재함. $\boldsymbol{h_t} = [h_{1,t}\ h_{2,t}\ \dots\ h_{m,t}]$

즉, 시점별 데이터를 전부 가지고 있는 "하나의 샘플"을,하나의 시퀀스(sequence)라고 말함.

RNN은 입력과 출력을 시퀀스 단위로 처리함. 은닉층 출력과 입력 데이터 간의 관계식은 다음과 같음.

**[ 행렬 표현 ]**

$$\boldsymbol{h_t} = f(\boldsymbol{x_t} \cdot \boldsymbol{W} + \boldsymbol{h_{t-1}} \cdot \boldsymbol{U} + \boldsymbol{b})$$

이때, $\boldsymbol{W}$는 $\boldsymbol{x}$와 $\boldsymbol{h}$의 관계를 나타내는 시불변 가중치 행렬.

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{d1} & \cdots & w_{dm} \end{bmatrix}$$

$\boldsymbol{U}$는 $\boldsymbol{h_{t-1}}$와 $\boldsymbol{h_t}$의 관계를 나타내는 시불변 가중치 행렬.

$$\boldsymbol{U} = \begin{bmatrix} u_{11} & \cdots & u_{1m} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mm} \end{bmatrix}$$

$\boldsymbol{b}$는 편향 벡터.

$$\boldsymbol{b} = [b_1\ b_2\ \dots\ b_m]$$

$f$는 활성화 함수(tanh 함수, 시그모이드 함수 같은 미리 정한 임의의 비선형 함수).

**[ 원소 단위 표현 ]**

$$[h_{1,t}\ h_{2,t}\ \dots\ h_{m,t}] = f\left( \begin{array}{c} [x_{1t}\ x_{2t}\ \dots\ x_{dt}] \cdot \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{d1} & \cdots & w_{dm} \end{bmatrix} \\ +[h_{1,t-1}\ h_{2,t-1}\ \dots\ h_{m,t-1}] \cdot \begin{bmatrix} u_{11} & \cdots & u_{1m} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mm} \end{bmatrix} \\ +[b_1\ b_2\ \dots\ b_m] \end{array} \right)$$

**[ 설명 ]**

RNN에서 중간의 $\boldsymbol{h_t}$를 계산할 때는, $\boldsymbol{x_t}$뿐 아니라 $\boldsymbol{h_{t-1}}$까지도 계산에 같이 넣는다. 이를 통해, 직전 timestep(시점)에서의 정보가 다음 timestep의 정보에도 영향을 줄 수 있게 된다.

수식에서 보듯이 $\boldsymbol{h_t}$의 $m$개 모든 노드는, $\boldsymbol{x_t}$의 $d$개 모든 노드와 $\boldsymbol{h_{t-1}}$(직전 시점)의 $m$개 모든 노드들에서 다 영향을 받는 구조다.

RNN 모델에 하나의 시퀀스 데이터가 들어오면, 0 시점부터 $T$ 시점까지 신경망 계산이 timestep번 만큼 반복해서 순환한다.

**[파라미터 수]**

신경망 학습은, 역전파 과정을 통해서, 최적의 $\boldsymbol{W}$($d \times m$ 행렬)와 $\boldsymbol{U}$($m \times m$ 행렬)와 $\boldsymbol{b}$를 찾아가는 과정이다. 즉, 전체 학습 파라미터 수는 1) $\boldsymbol{W}$의 $d \times m$개, 2) $\boldsymbol{U}$의 $m \times m$개, 3) $\boldsymbol{b}$의 $m$개를 더한 값이다.

즉, $d \times m + m \times m + m$개다.

## C. RNN과 VAR 모델의 유사성

은닉층의 m개의 노드 수치들은, 1) 입력층의 d개의 노드 수치와 은닉층의 직전 timestep의 m개 노드 과거 값들에다가 가중치 행렬을 곱하고 더해서 나온 수치에 2) 마지막 f 함수를 덧붙여 계산해 수치를 만들어낸다. 즉, 이전 항을 계산에 포함함으로써, 시계열 상의 변화를 표현할 수 있다. 이는 전통적인 VAR(Vector Autoregression) 모델과 유사성이 있음을 보인다.

우선, VAR 모델의 구조는 다음과 같다.

**VAR 모델**

$$\boldsymbol{y_t} = \boldsymbol{c} + \boldsymbol{A_1 y_{t-1}} + \boldsymbol{A_2 y_{t-2}} + \cdots + \boldsymbol{A_p y_{t-p}} + \boldsymbol{e_t}$$

이때, $\boldsymbol{y_t}$는 k 차원 벡터의 t 시점 값을 말함(쉽게 말해 k개 경제지표의 t 시점의 값 모음).

$\boldsymbol{A_i}$는 시불변 (k × k)-행렬, $\boldsymbol{e_t}$는 오류 항의 k- 벡터임.

즉, VAR는, k개 경제지표의 t 시점의 값이, 그 경제지표들의 t-1 시점, t-2 시점, ⋯, t-p 시점의 수치의 조합으로 결정된다는 접근법임.

이제 VAR(1) 모델을 생각해 보자.

$$\boldsymbol{VAR(1)}: \boldsymbol{y_t} = \boldsymbol{c} + \boldsymbol{A_1 y_{t-1}} + \boldsymbol{e_t}$$

$\boldsymbol{y_t}$는 k 차원 벡터의 t 시점

그리고, VAR와 유사한 RNN 모델을 한번 가정해 보자. RNN 모델 중에서, 1) 은닉층이 k 차원이고, 2) 입력 데이터가 아예 없고, 3) 활성화 함수가 단순 항등함수($f(x) = x$)인 특수하고 간단한 RNN 모델을 구상하면 다음과 같다.

$$\boldsymbol{h_t = h_{t-1} \cdot U + b}$$

$\boldsymbol{h_t}$는 k 차원의 벡터임

위의 VAR(1)과 이 특수 RNN을 보면, 둘은 오차항 부분을 제외하면 거의 같은 수식이라고 할 수 있다. 즉, VAR(1) 모델은 RNN 모델의 간단한 특수 케이스라고도 볼 수 있다.

VAR(p) 모델에서도 마찬가지다.

$$\boldsymbol{VAR(p): y_t = c + A_1 y_{t-1} + A_2 y_{t-2} + \cdots + A_p y_{t-p} + e_t}$$

$\boldsymbol{y_t}$는 k 차원 벡터의 t 시점

그리고, VAR(p)에 대응되는 RNN 모델을 생각하면, 1) 입력 데이터가 아예 없고, 2) 활성화 함수가 단순 항등함수($f(x) = x$)이면서 아래의 기준을 가지는 RNN 모델을 구상해 볼 수 있다.

$$\boldsymbol{h_t = [y_t\ y_{t-1} \cdots\ y_{t-p+1}]}$$

이때 $\boldsymbol{h_t}$는, "$y_t$와 과거 시점 변수"들을 옆으로 붙인 k*p 차원의 벡터임.

$$\boldsymbol{h_t = h_{t-1} \cdot U + b}$$

이 경우, RNN 모델의 표현식은 위 식과 같다. 여기서 $\boldsymbol{h_t}$ 내부에 $\boldsymbol{y_k}$들이 포함되어 있기 때문에, 위의 RNN 모델은 VAR(p)와 유사한 역할을 한다. 즉, VAR(p) 모델이 할 수 있는 일을, RNN 모델과 충분한 데이터만 있으면 비슷하게 수행할 수 있다고 할 수 있다.

정리하면, RNN 모델이, 과거의 시계열 계량 경제 모델이 하던 역할과 또 그 이상으로 복잡한 패턴 분석의 작업도 수행 가능하다고 말할 수 있다.

### 2) RNN 모델의 데이터 구조

일반적인 keras에서의 RNN 모델의 데이터 구조를 확인하는 것도, RNN 모델의 이해에 도움이 된다.

RNN 모델에서 X(독립변수, feature) 데이터는 보통 [n_samples, timesteps, n_features]의 3차원 텐서 형태다. (원래, 텐서는 모든 n차원 배열을 다 지칭하는 용어임. 특별히, 0차원은 스칼라, 1차원은 벡터, 2차원은 행렬이란 용어도 같이 사용함. 딥러닝에서는 보통 3차원 이상의 배열을 텐서라고 지칭하는 경우가 많음. 참고로, 구글의 TensorFlow는 다차원 데이터(tensor)가 계산 그래프를 흐르며(flow) 처리되는 프레임워크란 뜻에서 유래했음.)

RNN의 X 데이터에서 1개 샘플 데이터는, [timesteps, n_features] 형태의 수치 정보를 담고 있다. 이는, 정확히는 서로 다른 n_features 개의 데이터(앞의 "B. RNN의 수식 표현" 문단 안에서의 d 차원 벡터 $\boldsymbol{x}$)를 timesteps 수만큼의 시계열 정보로 가지고 있는 정보를 말한다. 이 데이터를 RNN 모델에 집어넣으면, timestep=0 시점부터 T 시점까지 데이터가 순차적으로 들어가면서, $\boldsymbol{x_i}$와 $\boldsymbol{h_i}$들이 하나씩 계산되는($\boldsymbol{h_{i-1}}$도 참고하면서) 방식이다.

### A. RNN 입력 데이터 예시 1

특정 주식의 "과거 주가와 거래량"이란 2개 정보를 사용해서 미래 주가를 예측하는 RNN 모델을 만든다고 가정해보자. 이때 시계열적인 연관성을 추적하기 위해, 일별 기준의 직전 20일 시계열 정보를 사용한다고 하자.

그러면, 모델 학습에 사용하는 X 데이터는 [n_samples, 20, 2]의 구조가 되어야 한다. 이런 X 데이터의 예시를 만들어서 출력하면 다음의 화면과 같이 출력된다.

```
# 예시 1의 데이터 샘플

> X.shape
(4888, 20, 2)

> print(X)
[[[   13180. 20955550.]
  [   13300. 17900900.]
  [   13980. 39290600.]
  ...
  [   14280. 29380700.]
  [   14320. 19647700.]
  [   14400. 16913900.]]

 [[   13300. 17900900.]
  [   13980. 39290600.]
  [   13580. 35518648.]
  ...
  [   14320. 19647700.]
  [   14400. 16913900.]
  [   14800. 23008650.]]

 ...

 [[  102000. 22169970.]
  [   99500. 20002282.]
  [  100500. 20899788.]
  ...
  [   96500. 18993346.]
  [  100600. 25360436.]
  [   94800. 23103574.]]]
```

print(X) 출력 결과에서, 맨 처음에 [ 13180. 20955550.]의 2개 수치 한 쌍이 나오는데 이는 "첫 번째 샘플의 시계열 상 첫째 날짜의 주가와 거래량 정보"를 말한다. 즉, 첫 번째 샘플 안에서의 $\boldsymbol{x_0}$를 의미한다.

그다음 [ 13300. 17900900.] 쌍은 "첫 번째 샘플의 시계열 상 둘째 날짜의 주가와 거래량 정보"를 의미한다. 즉, 첫 번째 샘플 안에서의 $\boldsymbol{x_1}$다. 이렇게 20개의 쌍이 모이면, 첫 번째 샘플 한 개에서의 전체 시계열 정보를 이룬다. 이 예제에서는 이런 샘플들을 4,888개 모아 놓았다.

RNN 모델의 학습 시에는, 한 개 샘플의 시계열 정보를 순차적으로 넣어서 시간순으로 $\boldsymbol{h_t}$들과 $\mathbf{y_T}$를 만들면서 역전파 과정으로 가중치 학습을 진행한다. RNN 모델의 예측 시에도 현재 샘플의 시계열 정보를 순차적으로 넣어서 시간순으로 $\boldsymbol{h_t}$들과 $\mathbf{y_T}$를 만든다(실제로는 미니 배치 방식으로 작동).

이제, 이 예시 데이터와 RNN 모델의 은닉층 노드 수를 8개로 했을 때의, RNN 계층 계산 수식을 예시로 정리해 보자.

**예시 1의 RNN 계산 수식**

**[ 가정 ]**

입력 데이터가 2차원의 벡터임. $\boldsymbol{x} = [x_{주가}\ x_{거래량}]$.

이때 $t$ 시점별로 데이터가 존재함. $\boldsymbol{x_t} = [x_{주가t}\ x_{거래량t}]$. $t$ 시점은 총 20개임.

은닉층 출력 데이터가 8차원의 벡터로 정함. $\boldsymbol{h} = [h_1\ h_2\ \dots\ h_8]$

이때 $t$ 시점별로 데이터가 존재함. $\boldsymbol{h_t} = [h_{1,t}\ h_{2,t}\ \dots\ h_{8,t}]$

은닉층 출력 데이터와 입력 데이터 간의 관계식은 다음과 같음.

**[ 행렬 표현 ]**

$$\boldsymbol{h_t} = f(\boldsymbol{x_t} \cdot \boldsymbol{W} + \boldsymbol{h_{t-1}} \cdot \boldsymbol{U} + \boldsymbol{b})$$

이때, $\boldsymbol{W}$는 $\boldsymbol{x}$와 $\boldsymbol{h}$의 관계를 나타내는 시불변 가중치 행렬임(2x8 행렬).

$$\boldsymbol{W} = \begin{bmatrix} w_{주가,1} & \cdots & w_{주가,8} \\ w_{거래량,1} & \cdots & w_{거래량,8} \end{bmatrix}$$

$\boldsymbol{U}$는 $\boldsymbol{h_{t-1}}$와 $\boldsymbol{h_t}$의 관계를 나타내는 시불변 가중치 행렬임(8x8 행렬).

$$\boldsymbol{U} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,8} \\ \vdots & \ddots & \vdots \\ u_{8,1} & \cdots & u_{8,8} \end{bmatrix}$$

$\boldsymbol{b}$는 편향 벡터.

$$\boldsymbol{b} = [b_1\ b_2\ \dots\ b_8]$$

$f$는 활성화 함수.

**[ 상세 표현 ]**

$$[h_{1,t}\ h_{2,t}\ \dots\ h_{8,t}] = f\left([x_{주가t}\ x_{거래량t}] \cdot \boldsymbol{W} + [h_{1,t-1}\ h_{2,t-1}\ \dots\ h_{8,t-1}] \cdot \boldsymbol{U}\ + \boldsymbol{b}\right)$$

**[ 계산 방법 ]** t 시점을 0~19까지 20번 진행하면서, 매번 $\boldsymbol{h_t}$를 만들어 냄.

**[파라미터 수 ]** 2*8 + 8*8 + 8 = 88개임.

다시 정리하면, 일반적인 RNN 모델에서 X 데이터의 형태는 [n_samples, timesteps, n_features]의 3차원 텐서다. 퀀트 투자 모델이라고 가정하면, 이때 n_features 개수만큼의 투자 판단 지표들이 사용되며, timesteps 개수만큼 이 지표들의 과거 관찰 기간 시계열 자료가 들어간다.

이런 X 데이터 구조는, 직전에 보았던 Conv1D 형태의 CNN 모델의 데이터 구조와 동일하다고 하겠다.

### * RNN, CNN, MLP의 파라미터 수 비교

완전한 일대일 비교는 아니지만, 일반적인 형태로 RNN, CNN, MLP 신경망을 쓸 때 파라미터 수의 차이를 비교하는 것도 의미가 있는 작업이다.

이때, 모델들의 공통 입력 데이터는 F개의 데이터의 T기간 동안의 시계열 정보라고 가정하자. 즉 데이터 구조는 [n_samples, T, F] 형태가 된다. 여기에서 각 모델들의 파라미터 수는 다음과 같이 계산된다.

**1) 노드가 M개인 RNN:** 파라미터 수가 (F + M + 1) * M이다.

**2) 필터 수가 M개고 kernel_size가 K(3 또는 5의 작은 수)인 CNN:** 파라미터 수가 (K * F + 1) * M이다.

**3) 노드가 M개인 완전 연결층 MLP:**
F개의 데이터의 T기간 동안의 시계열 정보 전체를 다음 계층과 완전 연결할 때는, F * T개 노드의 쭉 나열한 층으로 만든 다음 연결하는 방식임. 이 경우, 파라미터 수는 (F * T + 1) * M이다.

이때 CNN에서의 K는 보통 3 또는 5 정도의 작은 수를 쓴다. 따라서, 3개 모델의 파라미터 수를 비교하면, 세 변수가 곱해지는 MLP의 파라미터 수가 가장 많게 된다. RNN과 CNN의 파라미터 수는 상대적으로 작다.

RNN은 시계열 데이터를 t-1 시점과 t 시점의 관계식으로 만들고 재귀 방식을 도입하면서, 효율적으로 파라미터 수를 줄이는 방식이다. CNN은 국소 패턴 감지라는 합성곱 연산을 도입하여 효율적으로 파라미터 수를 줄인다.

## B. RNN 입력 데이터 예시 2

Simple RNN, LSTM 등 RNN 계열은 텍스트 분석, 자연어 처리에도 많이 활용된다. 하나의 텍스트 문장은 여러 토큰들이 순차적으로 배치되는 순차 데이터의 형태를 가진다(토큰은 의미를 가지는 최소 단위를 말함. 단어와 비슷한 개념). 즉, 셰익스피어 희곡에 나온 문장에서 앞의 20개 단어를 X, 21번째 단어를 y로 만들어서, 엄청나게 많은 데이터를 학습시킨다면, 그 모델은 주어진 문장에 잘 맞는 셰익스피어식 다음 단어를 붙여나갈 수 있다.

텍스트 단어는, 사실 컴퓨터로 수치화하기 상당히 어려운 정보다. 보통의 경우, 한 단어가 긴 차원의 벡터 하나로 저장된다(즉, 한 단어 자체가, 여러 feature를 가지는 긴 벡터를 써야만 표현이 가능하다고 할 수 있음). 예를 들어, 1) 각 단어들을 64차원 벡터로 저장한다고 하고, 2) 한 문장이 최대 100단어까지 가지도록 하고, 3) 이런 문장을 2만 개 샘플을 모았다고 가정해 보자. 이 중 하나의 샘플이 "I work hard"란 문장을 담고 있다고 하면, 이는 [1, 3, 64] 형태의 데이터로 저장된다(문장의 단어 개수를 100으로 맞추면, padding 처리하여 [1, 100, 64] 구조가 됨). 이런 샘플 데이터 전체는 최종 [20000, 100, 64] 구조의 3차원 텐서 데이터가 저장된다.

RNN 모델의 학습 혹은 예측 시에는, 각각의 시퀀스(각각의 한 개 샘플, 혹은 각 문장) 안의 단어들이 순차적으로 모델에 들어가면서 문장 내 단어의 전후 관계를 익히거나 혹은 예측에 활용되는 방식이다.

여기서의 텍스트 데이터나, 앞에서 본 금융 시계열 데이터나, 모두 [n_samples, T, F]의 3차원 텐서 형태의 데이터가 입력 데이터로 사용된다.

```
# 예시 2의 데이터 샘플

> X_embedding.shape
(20000, 100, 64)

> print(X_embedding)
[[[ 0.7862 -0.8304 -0.0191 ... -0.7724 -0.2566  0.4985]
  [-0.687   0.7983  0.5279 ... -0.4134 -0.1171  0.3751]
  [-1.4175 -0.0703 -0.3776 ... -0.3627 -0.9861 -1.6565]
  ...
  [ 0.5774 -0.9118 -0.0265 ...  0.2482  0.4753  0.2728]
  [-1.5296 -1.2601 -0.8696 ...  0.15    0.2018  0.6186]
  [-1.2683 -1.1843  1.1967 ...  0.5354 -0.1719  0.699 ]]

 [[-0.2456 -0.3242 -1.1257 ... -0.8066  0.9147 -1.875 ]
  [ 0.5298  2.0014 -0.8087 ...  0.5052  1.9802  0.217 ]
  [ 0.7487 -0.3558 -0.2242 ... -0.5396  0.4561 -0.708 ]
  ...
  [-0.6558  2.2142 -1.069  ... -0.2875 -0.4922 -0.6098]
  [-0.2029 -0.6872 -0.2817 ...  0.0504 -1.1927 -0.6905]
  [-0.3395  2.1164 -1.2726 ...  0.2236  0.8454  0.4387]]

 ...

 [[ 0.0226  1.5005 -0.8791 ...  0.5626  0.6115  0.0705]
  [ 1.2356 -0.2162  1.4151 ...  0.7877  1.5397  0.1614]
  [ 2.5226 -0.3474 -1.4424 ...  1.8566 -0.2955  0.2693]
  ...
  [-0.274  -0.5644  0.4541 ...  1.3024 -1.8674  1.0254]
  [ 0.4276  0.6406 -0.7073 ...  1.0075 -0.708   0.1376]
  [-0.1616  0.5288  2.2089 ... -0.603   0.7202  0.1549]]]
```

예시) 첫 줄 [0.7862 -0.8304 -0.0191 ... -0.7724 -0.2566 0.4985]가 첫 번째 샘플(문장)의 첫 번째 단어를 벡터로 표현한 것임.
둘째 줄 [-0.687 0.7983 0.5279 ... -0.4134 -0.1171 0.3751]가 첫 번째 샘플의 두 번째 단어를 의미함. 이렇게 100개 단어가 모여 첫 번째 문장을 구성함.
이런 문장이 2만 개가 모여서, 전체 X 데이터를 구성함.

# II. RNN 사용법과 예시 모델

## 1) RNN 사용법

keras에서는 Simple RNN 계층에 대해서 keras.layers.SimpleRNN 클래스를 제공한다.

SimpleRNN의 선언 문법은 다음과 같다.

```
keras.layers.SimpleRNN(
    units,
    activation=tanh,
    return_sequences=False,
    ***
)
```

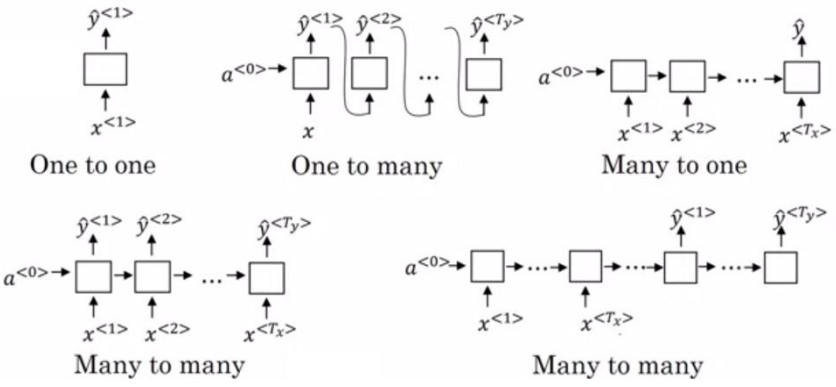
SimpleRNN 계층 선언 시, units는 RNN의 노드 수를 지정하는 변수다. 노드 수를 늘릴수록 다양한 시계열 상태 정보를 각각 저장하므로 복잡한 패턴을 탐지할 수도 있다. 그러나, 그만큼 과적합의 위험도 늘어나는 단점도 있다. 보통 2의 거듭제곱인 16, 32, 64 등의 수치를 많이 사용한다.

SimpleRNN도 다른 계층처럼 당연히 활성화 함수를 가진다. 순환 신경망은 그 특성상 반복 계산에 의한 수치 불안정 가능성이 높아서, 활성화 함수(activation 옵션)로 tanh 함수를 주로 쓴다.

return_sequences 옵션은 RNN 레이어의 출력을 1) 마지막 timestep의 은닉층 노드 값($\boldsymbol{h_T}$)만 할지, 2) 모든 timestep의 은닉층 노드 값($\boldsymbol{h_t}$)을 모두 출력할지를 정하는 옵션이다.

사실 RNN 층의 출력은 여러 가지 형태의 출력이 다 가능하다(신경망 세계에서는, 상상 가능한 매우 다양한 네트워크 구조가 다 연구되고 만들어지고 있음). 1) 마지막 timestep의 은닉층 노드만 출력하는 것을 many-to-one 구조라고 하고, 2) 모든 timestep의 은닉층 노드를 다 출력하는 것을 many-to-many 구조라고 한다.

**RNN 모델 입출력의 다양한 구조**




자료: https://iq.opengenus.org/types-of-rnn/

## A. many-to-one 구조

하나의 시퀀스를 받아서 RNN이 시점(timestep)별 반복 계산을 할 때, 마지막 timestep의 은닉층 노드 정보만 다음 계층으로 출력을 보내는 방식이다. 즉, T 번 계산한 다음 $\boldsymbol{h_T}$ 벡터만을 다음 층으로 전달한다.

예를 들어, 주식의 과거 주가와 거래량 정보의 20일 치를 받아서 다음날 주가를 예측하는 RNN 모델을 만들려고 한다면, 이런 many-to-one 구조가 사용된다.

RNN의 입력 데이터 shape를 [n_samples, timesteps, F]로 하고, RNN의 unit 개수를 M으로 지정했다고 하자. 이 RNN을 many-to-one 구조를 만들면, RNN은 각 샘플별로 timesteps만큼 순회 처리를 한 후 마지막 timestep의 노드 정보만 출력한다. 따라서, 이 출력 데이터의 shape는 [n_samples, M]이 된다.

SimpleRNN 선언 시 many-to-one 구조로 만들려면, return_sequences=False의 디폴트 그대로 쓰면 된다.

## B. many-to-many 구조

하나의 시퀀스를 받아서 RNN이 시점(timestep)별 반복 계산을 할 때, 모든 timestep의 은닉층 노드를 다 출력한다. T 번 계산할 동안 $\boldsymbol{h_i}$ 벡터를 다음 층으로 매번 전달한다.

예를 들어, 시계열을 시계열로 전환하는 모델을 만들거나 RNN 층을 여러 층을 쌓을 때 중간층의 구성법에서 이런 many-to-many 구조가 사용된다.

RNN의 입력 데이터 shape를 [n_samples, timesteps, F]로 하고, RNN의 unit 개수를 M으로 지정했다고 하자. 이 RNN을 many-to-many 구조를 만들면, RNN은 각 샘플별로 timesteps만큼 순회 처리하는 동안 각 timestep에서의 은닉층 노드를 계속 출력한다. 따라서, 이 출력 데이터의 shape는 [n_samples, timesteps, M]이 된다.

SimpleRNN 선언 시 many-to-many 구조로 만들려면, return_sequences=True로 지정하면 된다.

**many-to-one과 many-to-many의 비교**

| 입출력 방식 | many-to-one | many-to-many |
|---|---|---|
| 입력 데이터 구조 | [n_samples, timesteps, F] | |
| RNN층 노드 수 | M 개 | |
| 학습 파라미터 수 | (F + M + 1) * M | |
| 출력 데이터 구조 | [n_samples, M] | [n_samples, timesteps, M] |

### 2) 예시 모델

RNN 모델의 사용법을 익히기 위해서, 사인 파동을 데이터만으로 학습하고 예측하는 예시 모델을 소개한다. 사인 함수 데이터는 직전값들의 정보가 다음 값과 연관을 가지는 시계열 모델로 근사가 가능한 케이스다.

### A. 모델 구상

사인 파동의 시계열 데이터를 가지고, 과거 일정 기간의 시계열 정보를 넣고, 바로 다음 시점의 수치를 예측하는 목적의 RNN 모델을 만든다. 이때, 시계열 예측 시, 성과 비교 대상 모델을 같이 추가하는 것이 좋다. 시계열 자료에서 가장 초보적인 예측은, 직전 수치가 그대로 유지될 것으로 믿고 직전값을 그대로 쓰는 방식일 것이다. 이를 naïve 예측으로 명명했으며, RNN 모델과 naïve 예측의 성과를 서로 비교했다.

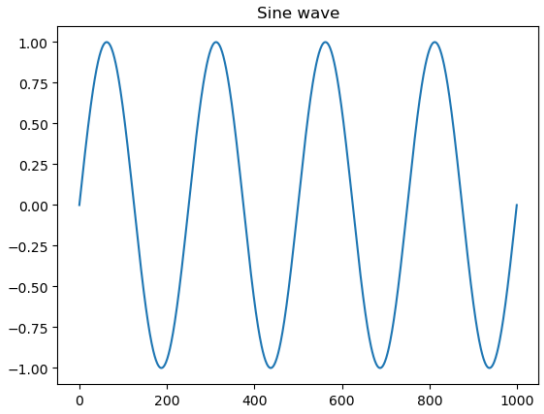
### B. 데이터 생성 및 RNN 모델 선언 함수 코드

```
import numpy as np
import matplotlib.pyplot as plt

N = 1000
LOOKBACK = 100

# ------------------------------------------------------------
# 데이터 생성 함수
# ------------------------------------------------------------
def make_data(n, lookback):

    # 사인파 생성
    t = np.linspace(0, np.pi * 8, n)
    sin_wave = np.sin(t)

    plt.plot(sin_wave)
    plt.title("Sine wave")
    plt.show()

    # ------------------------------
    # X, y 생성
    #    입력: t0 이전 lookback 구간의 수치들
    #    타깃: t0 시점의 수치
    # ------------------------------
    X_list, y_list = [], []
    for i in range(lookback, n):
        X_list.append(sin_wave[i - lookback:i])
        y_list.append(sin_wave[i])

    X = np.array(X_list).reshape(-1, lookback, 1) # (samples, timesteps, 1)
    y = np.array(y_list)                          # (samples,)

    return X, y
```

위 코드에서 우선, 사인 파동이 4번 반복되는 길이 1,000의 시계열 변수(sin_wave)를 만들었다.
그 다음에, RNN에서 쓸 길이(lookback)가 100인 시계열 학습 데이터를 만들기 위해서, 슬라이딩 윈도우 방식 형태로 "t0 시점 이전의 lookback 길이의 수치 시계열을 X로 지정하고, y0 시점 수치를 y로 지정하는" 작업을 수행했다. 이런 슬라이딩 윈도우 생성법은, 직전 "(7) CNN AI 투자 모델 (기술적 분석 기반)" 리포트에서와 동일한 방식이다.

메인 코드 내에서 다음을 실행하면, 원시 데이터(sin_wave)가 출력되고, RNN 모델용 X와 y 데이터가 생성된다.

```
> X, y = make_data(n=N, lookback=LOOKBACK)
```




```
> X.shape
(900, 100, 1)

> y.shape
(900,)
```

다음은 RNN 모델 선언 함수 코드다.

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

UNITS = 16

# ------------------------------------------------------------
# RNN 모델 정의 함수
# ------------------------------------------------------------
def build_rnn_model(input_shape, units):

    model = keras.Sequential([
        layers.Input(shape=input_shape),
        layers.SimpleRNN(units),
        layers.Dense(1)
    ])

    model.compile(optimizer=keras.optimizers.Adam(1e-3), loss="mse", metrics=["mse"])
    model.summary()

    return model
```

**RNN 모델의 네트워크 구성도**

0. 입력 데이터: 사인 파동의 직전 시계열 데이터(100개 시점 사용) (shape=[N, 100, 1]) →

1. RNN 계층 (유닛 수 16개) →

2. Output(Dense) 계층: 사인 파동의 바로 다음 수치. 데이터 shape는 [N,]임

네트워크 구성은, 먼저 "각 timestep별 사인파 변화를 담고 있는 데이터"가 입력층으로 들어온다. 다음에는 16개 유닛의 RNN 은닉층이 놓인다. 다음에는 노드가 1개인 출력층(dense 층)이 놓인다. 여기에 사인 파동의 바로 다음 수치(y)가 들어간다.

메인 코드 내에서 함수 작동 코드를 실행하면, 다음과 같이 model.summary()의 결과가 나온다.

```
> model = build_rnn_model(input_shape=X.shape[1:], units=UNITS)

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_2 (SimpleRNN)    (None, 16)                288

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 305 (1.19 KB)
Trainable params: 305 (1.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

이 모델에서의 파라미터 수도 확인해 보자. RNN 직전 층에서는 한 개 샘플의 구조가 [100, 1]이다. RNN 층에서는 유닛이 16개가 쓰인다. 따라서, RNN 층의 파라미터 수는 (1 + 16 + 1) * 16 = 288이다. RNN 다음의 출력층은 노드가 1개다. RNN의 16개 노드 층과 1개 노드 층이 완전 연결하기 위해서는, 파라미터가 16 * 1 + 1 * 1이 쓰여서, 17개가 된다. 따라서, 전체 신경망 모델에서 전체 학습 파라미터 수는 이 둘을 합친 305개가 된다.

사인파는 원래 2차 선형 점화식으로 표현이 가능하다. 따라서, 샘플 데이터 수를 엄청 크게 하고, 활성화 함수를 맞게 수정하고, 계속 반복해서 학습을 하면, 이론상으로는 RNN의 노드가 단 2개만 있더라도 사인파 예측 모델을 만들 수가 있다.

하지만, 여러 현실적 제약으로 인해, RNN 층의 노드 수를 적당히 늘리는 것이 학습이 더 잘 되는 편이다. 여기서도 노드 수를 16개 정도로 적당히 늘려서 사용했다.

## C. 메인(학습 및 평가) 코드

```
# ------------------------------------------------------------
# MAIN
# ------------------------------------------------------------
if __name__ == "__main__":

    # ----------------------------
    # 1. 데이터 생성
    # ----------------------------
    X, y = make_data(n=N, lookback=LOOKBACK)

    # ----------------------------
    # 2. 데이터 분할
    # ----------------------------
    # 시계열이므로 섞지 않고 train/val/test 분할
    X_train, X_raw, y_train, y_raw = train_test_split(X, y, test_size=0.5, shuffle=False)
    X_val, X_test, y_val, y_test   = train_test_split(X_raw, y_raw, test_size=0.5, shuffle=False)

    # ----------------------------
    # 3. 모델 정의
    # ----------------------------
    # 모델 선언
    model = build_rnn_model(input_shape=X.shape[1:], units=UNITS)
    # 콜백 정의
    early_stopping_cb = EarlyStopping(patience=10, restore_best_weights=True)
    reduce_lr_cb = ReduceLROnPlateau(factor=0.5, patience=5, min_lr=0.00001)
    callbacks = [early_stopping_cb, reduce_lr_cb]

    # ----------------------------
    # 4. 모델 학습
    # ----------------------------
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=200,
        callbacks=callbacks,
        verbose=0,
    )

    # ----------------------------
    # 5. 학습 결과 출력
    # ----------------------------
    plt.plot(history.history["loss"], label="loss")
    plt.plot(history.history["val_loss"], label="val_loss")
    plt.title("Model loss")
    plt.legend()
    plt.show()

    # ----------------------------
    # 6. 테스트 성능 평가
    # ----------------------------
    # RNN 예측 성능
    pred_rnn = model.predict(X_test, verbose=0).flatten()
    mse_rnn = np.mean((y_test - pred_rnn) ** 2)

    # naive 예측 성능: 마지막 timestep 수치 그대로 사용
    pred_naive = X_test[:, -1, 0]
    mse_naive = np.mean((y_test - pred_naive) ** 2)
```

```
# 성능 비교
print(f"MSE_RNN   = {mse_rnn:.6f}")
print(f"MSE_Naive = {mse_naive:.6f}")
print(f"Improvement (MSE_Naive/MSE_RNN) = {mse_naive / mse_rnn:.2f}x")

# ------------------------------
# 7. 시각화: test 일부 구간 비교
# ------------------------------
plt.plot(y_test[:25], label="True")
plt.plot(pred_rnn[:25], label="RNN")
plt.plot(pred_naive[:25], label="Naive (last point)")
plt.xticks([])
plt.title("Next-step prediction on test")
plt.xlabel("Test time index")
plt.ylabel("Value")
plt.legend()
plt.show()
```

전체 메인 실행 코드는 위에 나온 것과 같다.

"1. 데이터 생성" 파트는 X와 y 데이터를 만들고, "2. 데이터 분할"에서는 이를 2:1:1의 비율로 각각 train, valid, test 데이터로 만든다. "3. 모델 정의"에서 모델 선언을 진행하고, "4. 모델 학습"에서는 model.fit을 실행한다. "5. 학습 결과 출력"에서는 학습 시 각 epoch마다 측정된 train 데이터 기준의 손실값과 val 데이터 기준의 손실값 추이를 그래프로 보여준다. 여기까지의 코드는, "TimeSeriesSplit 교차 검증이 아닌 하나의 train 및 val 데이터 세트를 쓴다는" 차이점만 빼면, 직전 리포트들에서 나왔던 코드와 거의 유사하다고 하겠다.

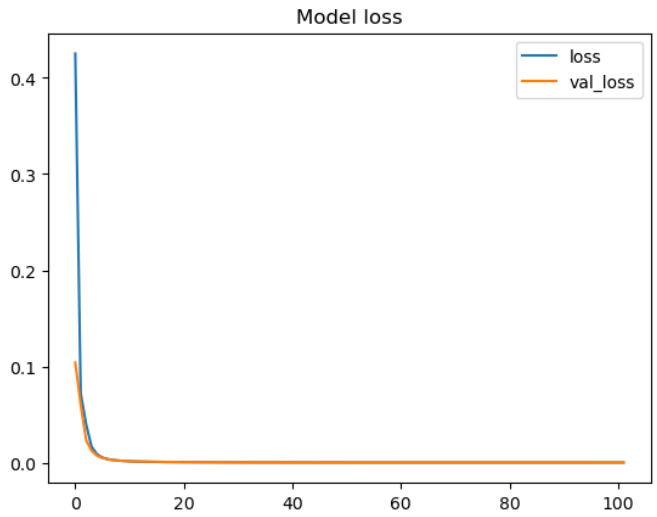
"6. 테스트 성능 평가"에서는 별도 test 데이터 세트를 사용해, RNN 모델의 예측 성능과 naïve 예측 성능을 비교하는 작업을 진행했다.

이 코드에서 y_test의 shape는 (225,)이고, model.predict(X_test) 예측 결과의 shape는 (225, 1)로 나온다. 즉, shape 형태가 다르게 나온다. 따라서, 정확한 MSE 계산을 위해서는, model.predict(X_test).flatten() 코드를 통해서 shape 변경을 꼭 해야 한다는 점을 기억할 필요가 있다.

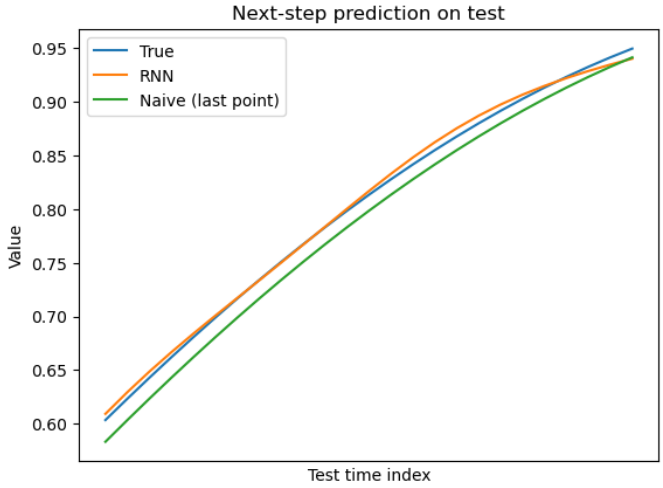
한편, 시계열 예측 작업에서, 내일의 수치를 예측할 때 오늘의 수치를 똑같이 대입하더라도 그래프상에서는 대충 그럴듯해 보일 수 있다. 이런 기초적인 오류를 피하기 위해서, 새로 개발한 모델과 "아주 단순한 가정의 naïve 예측법"과의 성능를 수치로 비교하는 과정이 있는 것이 유용하다. 여기서의 naïve 예측법은, 마지막 시점 값을 그대로 예측값으로 쓰는 방식을 사용했다. "6. 테스트 성능 평가"에서는 이 둘 간의 MSE 차이를 수치로 비교한다.

"7. 시각화"는 RNN 모델 예측 결과와 naïve 예측 결과를 그래프로 보여주는 역할을 한다.

이상의 메인 코드를 실행하면, 그 결과는 다음처럼 출력된다(4번 단계 이후만 표시함).




```
MSE_RNN   = 0.000101
MSE_Naive = 0.000290
Improvement (MSE_Naive/MSE_RNN) = 2.87x
```




우선 첫 번째 학습 결과 그래프를 보면, epoch가 진행되면서 손실값이 떨어지고 있음을 볼 수 있다.

다음의 텍스트 출력을 보면, test 데이터 세트를 가지고 RNN 모델과 Naïve 예측의 성능을 비교했을 때 RNN의 MSE가 0.0001, Naïve의 MSE 가 약 0.0003이어서 RNN의 MSE가 더 낮음을 알 수 있다. RNN을 사용한 사인파 예측 시 성능이 개선된다고 볼 수 있다(random_seed에 따라 수치 변동은 존재함).

마지막 그래프에서는 test 데이터 세트 일부 기준으로 실제 1) 그래프값(파란색), 2) RNN 예측값(주황색), 3) Naïve 예측값(녹색)을 같이 보여준다. 그래프상 RNN 예측값이 실제와 더 유사함을 보여준다. Naïve 예측값은 직전 시점 값을 쓰기 때문에 실제 값을 계속 후행해서 움직이고 있는 것을 그래프를 통해 알 수 있다.

## 3) RNN의 활용법

### RNN의 특징

RNN 계열의 모델은 입력 데이터에 시계열 자료를 그대로 넣을 수 있는 게 가장 큰 특징이다. 필자가 항상 예시로 드는 멘트가 "ISM 제조업 지수가 현재 50포인트이더라도, 48, 49, 50으로 올라온 패턴과 52, 51, 50으로 떨어진 패턴은 다르다"라는 문구다. 금융 관련 지표의 경우, 현재 수치가 얼마인지도 중요하지만, 이 수치가 최근까지 어떻게 흘러왔는지가 더 중요한 경우도 많다. 그래서, 당사의 DDQM2 모델에서도, 시장 및 경제지표의 현재 수치뿐 아니라 1개월 및 3개월 추세 정보를 추가로 고려하는 방식을 쓰는 것도 이런 이유다.

RNN 모델을 쓰게 되면, 이런 시계열 조작과 같은 별도의 피처 엔지니어링 대신에 원 데이터의 시계열 정보를 그대로 사용할 수 있다는 점에서 편리하다. 이 방식은 또 시계열 상에서의 미묘한 연관 관계를 찾는 데 효율적일 수 있다(단순히 1개월, 3개월 변화율만이 아닌, 그 중간의 유의미한 시계열 지점을 찾는 등의).

### RNN과 HMM(Hidden Markov Model), VAR(Vector Autoregression)

RNN은 기존의 전통적 계량 경제 모델을 대체, 보완하는 용도로도 가능하다. 대표적으로 HMM과의 비교가 가능하다. HMM 모델은 겉으로 보이는 관측값 뒤에 Hidden State가 존재하고 이 관계를 통해 국면 정의 및 예측을 하는 모델이다. RNN 모델도 기본적으로 시계열 연관성을 가지는 "은닉층 노드"를 도입하는 방식이므로, HMM과 비슷한 역할을 할 수 있다. 둘을 비교하면, 우선 HMM은 은닉 상태가 이산적이며, 은닉 상태가 확률적으로 변경되고, 이해 및 해석이 용이한 장점이 있다. 반면, RNN 계열 모델은 은닉 노드가 연속 벡터의 값을 가지는 게 특징이며, 비선형 패턴도 익힐 수 있는 장점이 있고, 또 시계열 연관성(장기 기억)을 가질 수 있는 장점이 있다. 만약 가용 데이터가 충분히 많다면, RNN이 더 효율적인 예측을 수행할 가능성이 높다고도 볼 수 있다.

HMM 외에도, 앞서 본 VAR(Vector Autoregression) 모델의 경우에도 RNN이 비슷한 역할의 모델링을 수행하는 것이 가능하다. 글로벌 투자 자산군 간의 전이 효과 같은 연관성을, RNN 모델 안에서 구현할 수도 있다.

### RNN과 CNN

RNN의 퀀트 모델 활용 시, CNN과 RNN을 결합하는 경우도 자주 존재한다. 쉽게 말하면 CNN은 "패턴 추출기"에 해당한다. 그리고, RNN은 "시간 누적기(시계열 연관 모델)"에 해당한다. 즉, CNN은 국소 패턴을 추출하여, 모멘텀 지표, 기술적 지표 등을 (유용하게) 자동으로 생성하는 기능을 담당한다. 그리고, RNN은 CNN이 뽑은 패턴을 가지고 누적 혹은 시계열 연관도를 만들어서, "이 패턴이 최근에 몇 번 나왔는지, 시계열 상으로 어떤 패턴으로 변했는지"를 탐지하는 역할을 한다. 이렇게 "CNN → RNN" 연결 방식으로 네트워크를 구성하여 퀀트 모델을 만드는 방법들도 활발히 연구되고 있다.

정리하면, RNN은 다양한 퀀트 모델에서 사용될 수 있다. 예를 들어, 1) 변동성 지수의 상위 N % 분포를 사용한 risk-on, risk-off 국면 탐지, 2) 글로벌 자산군의 전이 효과를 활용한 자산 가격 예측, 3) 팩터 과거 추이와 외부 영향 변수를 고려하는 팩터 타이밍 모델 개발 등의 작업에 RNN 모델들이 폭넓게 활용될 수 있다.

한편, RNN 계열 중 SimpleRNN은 장기 기억 소실의 이슈가 있어서, 최근에는 이를 보완한 LSTM 등의 모델이 더 많이 사용되는 편이다. RNN 계열의 실전적 모델링 관련해서는 향후 LSTM 편에서 다시 자세히 다룰 예정이다.

# V. Appendix: RNN 예시 모델의 전체 코드

```
# -*- coding: utf-8-sig -*-
"""사인파 데이터의 다음 step 예측에서, RNN 예측과 naive 예측(직전 값 그대로 예측)을 비교"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

N = 1000
LOOKBACK = 100
UNITS = 16

# ------------------------------------------------------------
# 데이터 생성 함수
# ------------------------------------------------------------
def make_data(n, lookback):

    # 사인파 생성
    t = np.linspace(0, np.pi * 8, n)
    sin_wave = np.sin(t)

    plt.plot(sin_wave)
    plt.title("Sine wave")
    plt.show()

    # ------------------------------
    # X, y 생성
    #    입력: t0 이전 lookback 구간의 수치들
    #    타깃: t0 시점의 수치
    # ------------------------------
    X_list, y_list = [], []
    for i in range(lookback, n):
        X_list.append(sin_wave[i - lookback:i])
        y_list.append(sin_wave[i])

    X = np.array(X_list).reshape(-1, lookback, 1) # (samples, timesteps, 1)
    y = np.array(y_list)                          # (samples,)

    return X, y


# ------------------------------------------------------------
# RNN 모델 정의 함수
# ------------------------------------------------------------
def build_rnn_model(input_shape, units):

    model = keras.Sequential([
        layers.Input(shape=input_shape),
        layers.SimpleRNN(units),
        layers.Dense(1)
    ])

    model.compile(optimizer=keras.optimizers.Adam(1e-3), loss="mse", metrics=["mse"])
    model.summary()

    return model
```

```
# ------------------------------------------------------------
# MAIN
# ------------------------------------------------------------
if __name__ == "__main__":

    # ------------------------------
    # 1. 데이터 생성
    # ------------------------------
    X, y = make_data(n=N, lookback=LOOKBACK)

    # ------------------------------
    # 2. 데이터 분할
    # ------------------------------
    # 시계열이므로 섞지 않고 train/val/test 분할
    X_train, X_raw, y_train, y_raw = train_test_split(X, y, test_size=0.5, shuffle=False)
    X_val, X_test, y_val, y_test   = train_test_split(X_raw, y_raw, test_size=0.5, shuffle=False)

    # ------------------------------
    # 3. 모델 정의
    # ------------------------------
    # 모델 선언
    model = build_rnn_model(input_shape=X.shape[1:], units=UNITS)
    # 콜백 정의
    early_stopping_cb = EarlyStopping(patience=10, restore_best_weights=True)
    reduce_lr_cb = ReduceLROnPlateau(factor=0.5, patience=5, min_lr=0.00001)
    callbacks = [early_stopping_cb, reduce_lr_cb]

    # ------------------------------
    # 4. 모델 학습
    # ------------------------------
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=200,
        callbacks=callbacks,
        verbose=0,
    )

    # ------------------------------
    # 5. 학습 결과 출력
    # ------------------------------
    plt.plot(history.history["loss"], label="loss")
    plt.plot(history.history["val_loss"], label="val_loss")
    plt.title("Model loss")
    plt.legend()
    plt.show()

    # ------------------------------
    # 6. 테스트 성능 평가
    # ------------------------------
    # RNN 예측 성능
    pred_rnn = model.predict(X_test, verbose=0).flatten()
    mse_rnn = np.mean((y_test - pred_rnn) ** 2)

    # naive 예측 성능: 마지막 timestep 수치 그대로 사용
    pred_naive = X_test[:, -1, 0]
    mse_naive = np.mean((y_test - pred_naive) ** 2)
```

```
# 성능 비교
print(f"MSE_RNN   = {mse_rnn:.6f}")
print(f"MSE_Naive = {mse_naive:.6f}")
print(f"Improvement (MSE_Naive/MSE_RNN) = {mse_naive / mse_rnn:.2f}x")

# ------------------------------
# 7. 시각화: test 일부 구간 비교
# ------------------------------
plt.plot(y_test[:25], label="True")
plt.plot(pred_rnn[:25], label="RNN")
plt.plot(pred_naive[:25], label="Naive (last point)")
plt.xticks([])
plt.title("Next-step prediction on test")
plt.xlabel("Test time index")
plt.ylabel("Value")
plt.legend()
plt.show()
```

# 삼성 All That Quant

**[팩터 모델 실전 전략 및 이론]**

25/10/27 "영업이익 증가율 모멘텀 팩터 - 신규 추천 전략" - High return, Low correlation의 고효율 팩터전략 신규 개발 및 제시
25/8/19 "Long-only 전략 상세 분석법 (롱 팩터 전략 시리즈 #2)" - 효율적 Long-biased 전략을 위한 프로세스와 추천 전략 제시
25/7/21 "Long-only 전략과 비대칭 알파 ((롱 팩터 전략 시리즈 #1)" - 코스피 5000 시대를 대비하는, long-biased 전략 집중 분석
24/11/20 "연구개발 팩터 투자전략" - R&D에 기반한 신규 팩터 전략을 개발하고, 이의 특징과 활용 방법 등을 제시함
24/8/8 "코스닥 투자자를 위한 퀀트 전략 Redux" - 코스닥 시장에서 변화하는 팩터 및 스타일 전략의 특징을 상세 재정리함
24/7/2 "연기금 투자자를 위한 장기 퀀트 전략" - 투자 horizon을 장기로 가져갔을 때의 팩터 투자 전략의 변화를 상세하게 비교 정리함
23/11/30 "퀀트 전략에서의 업종 특성 고려, 2편 - 업종 대비 상대 밸류에이션 팩터의 개선 전략" - 상대 밸류 팩터에서의 업종 특성 고려 전략 제시
23/9/4 "퀀트 전략에서의 업종 특성 고려, 1편 - 저P/B 팩터의 업종 조정 전략" - 한국 업종의 특성을 감안한 수정된 밸류에이션 전략 제시
*23/7/26 "Data-driven Quant Model 2 - #연속 국면 #머신러닝 #부스팅 기법 #동적 팩터 모델" - 부스팅 알고리즘 기반의 팩터 로테이션 모델 제시*
23/5/17 "퀄리티 팩터는 정말 퀄리티 있는 전략인가" - 퀄리티 팩터가 가지는 고유 특성과 세부 전략 간의 차이점을 비교 정리
23/1/4 "포트 전략 - 롱/숏 시그널로 롱 포트폴리오 구성하기" - 롱/숏 전략을 Long-only 포트로 전환하는 효율적인 알고리즘을 신규 제시
*22/11/21 "Data-driven Quant Model - 머신러닝 기법으로 시장의 파도에 올라타기" - 머신러닝 기법에 의한 데이터 기반의 팩터 로테이션 모델 제시*
22/8/17 "변동성 구간에서의 주식 스타일 특징 - 낙폭 과대주 팩터 활용법" - 팩터 투자 틀 안에서 낙폭 과대주 팩터의 필요성 입증과 활용법 제시
22/5/25 "경기 국면에 따른 리스크 팩터 배분 전략" - 리스크 팩터 전반에 대한 설명과 함께 팩터와 경기 국면을 결합한 투자 전략을 제시
21/2/24 "이익 모멘텀의 차별화" - 종목 특성을 감안한, 변동성 조정 이익 모멘텀 팩터 전략을 제시
21/1/11 "주식 스타일 Fact Book - 주식 스타일 및 매크로 변수 상관성 팩트 체크" - 주식 스타일 분석 체계의 정리 및 매크로 변수 연결
19/4/15 "팩터 모델의 구조와 해설 - Factor Models for Asset Returns" - 팩터 모델 이론에 대한 상세 설명 자료
18/7/3 "Fama-French 3 팩터 모델 - 후편" - Fama & French의 3팩터 모델 1993년 논문에 대한 전문 번역과 한국 시장 분석 추가 자료
18/5/8 "Fama-French 3 팩터 모델 - 전편" - Fama & French의 3팩터 모델 1992년 논문에 대한 전문 번역과 한국 시장 분석 추가 자료
15/10/20 "가치주 투자와 금리의 연관성" - 금리와 가치/성장 스타일의 인과 분석을, Equity Duration 개념을 활용하여 국내 최초로 제시
14/11/24 "투자 Horizon에 따른 퀀트전략 변화 - Advanced Factor Book" - 1M ~ 6M의 투자 기간별로 팩터 모델 백테스팅 분석
13/4/23 "매크로 기반 Dynamic Quant 모델" - 매크로 국면 판단에 기반한 동적 팩터 투자 모델 정립
12/2/27 "스타일 로테이션의 허와 실" - 스타일 로테이션 전략의 형태로서 모멘텀 방식, 리버전(컨트래리안) 방식, Asness 방식을 각기 비교 분석

**[퀀트 전략 일반]**

*26/1/8 "퀀트 모델링 A to Z - (7) CNN AI 투자 모델 (기술적 분석 기반)" - CNN 모델의 전체적인 이해와 사용 방법 등을 설명*
*26/1/2 "퀀트 모델링 A to Z - (6) MLP(Multi-Layer Perceptron) AI 투자 모델" - 딥러닝을 활용한 퀀트 투자 모델 개발법을 상세히 설명함*
25/11/14 "자산배분 전략의 체계" - 자산배분에 대한 기본적인 개념과 접근법을 체계적으로 정리함
*25/9/2 "머신 러닝 애널리스트, 파트 1" - 주식 시장의 기업 펀더멘털 분석에서, 머신러닝 및 AI를 활용하는 접근법 제시*
*25/7/29 " KR-FinBERT의 활용법" - 센티먼트 분석 AI를 손쉽게 사용하는 방법을 정리함*
25/5/20 "Corporate Action Book ver 2.0" - 여러 corporate action 이벤트에 대한 메커니즘 설명과 시장 반응의 최근 패턴을 정리함
25/3/20 "MSCI 지수 업데이트 ver 5.0" - MSCI 지수의 최근 이슈과 방법론 상의 변화, 정기 리뷰 로직의 요약 정리 제공
25/1/23 "MSCI 미국 지수 2월 리뷰 편출입 최종 전망" - 24년말부터 MSCI USA 종목 편출입 전망 리포트를 최초 개시함
25/1/14 "TAX 전략 2. 소득세뿐 아니라 건강보험료도 존재한다" - 개인 투자자 관점에서의 건강보험료 및 광의의 세금에 대한 설명 및 전략 자료
*24/11/7 "AI와 퀀트 투자" - AI에 의한 세상의 변화와 알고리즘에 대한 이해, 퀀트 투자와의 접목법을 정리함*
24/5/27 "KOSPI 지수 장기 수익률 성과 분해" - KOSPI 장기 수익률을 기업이익 증가, 주식수 변동, 밸류에이션으로 분해하여 부진 원인을 정리
24/5/8 "주당 지표(per share) 기준의 펀더멘털 접근법" - 지수 차원에서의 주당 지표 산출법과 이를 통한 시장 분석, 관련 팩터 전략 등을 정리
24/1/29 "이전 상장 이벤트 및 활용 전략" - 코스피 이전 상장 종목에 대한 주가 변화 결정 요인 상세 분석
23/7/11 "한미 간 자사주 매입의 구조적 차이" - 자사주 매입에 대한 한국과 미국 제도를 비교하고, 시가총액 정의 차이에 따른 문제점을 정리
23/4/17 "NSS(Nelson Siegel Svensson) 모델을 활용한 채권 퀀트 전략, 파트2" - NS 및 NSS 모델을 활용한 실전적인 채권 퀀트 전략 제시
22/11/7 "연방기금 선물 내재 금리의 이해" - 금리선물 등에 기반한 미래 시점별 정책금리 전망치를 확인하는 방법 및 이슈 정리
22/8/1 "주식과 채권의 상관성, 파트2 - 주식과 채권 간 상관 관계의 다이내믹스" - 주식/채권 상관관계의 변화를 경기과 물가 요인으로 설명 및 전망
20/10/28 "한국이 MSCI 선진시장에 들어가야 하는 이유" - MSCI 선진시장 승격의 당위성과 승격 시의 변화 등에 대해 총정리
*19/11/18 "코스피 단기예측 AI 모델 - 랜덤 포레스트 기법을 활용한 머신러닝 기반 모델" - 랜덤 포레스트 기반의 시장 예측 모델 및 소스 코드 제시*
17/3/8 "Volatility Drag - 레버리지 투자 수익과 변동성 손실의 관계" - 변동성 손실 개념과 레버리지 상품의 위험성에 대한 상세 설명
*16/1/4 "기계학습(Machine Learning)과 투자전략 - 빅 데이터 기법: Naive Bayes Classifier의 활용" - 머신 러닝에 기반한 퀀트 전략 개발*
15/7/20 "퀀트 Investment Clock, 한국편" - 성장과 물가 지표를 조합한 4개 국면 기반의 자산배분 모델 제시
14/4/24 "CAPE in Everywhere - Cyclically Adjusted P/E의 다용도 활용 분석" - 한국 증시의 CAPE 지표에 대한 본격 분석
11/5/17 "MSCI 종목편입 효과 실증분석 - 종목 편출입에 따른 최적 투자전략" - MSCI 이벤트 활용 투자 전략의 시작

## Appendix. Samsung Q.Pack 메일링 자료

글로벌 주식시장 동향부터, FICC, 경제지표, 국내외 종목 및 ETF 정보까지 수록함. 이메일을 통해 정기적으로 엑셀 자료 전송

| Index | Region /Country | Current Index (p) | Index change (%) (a=b+c) 1D | 2D | 1W | MTD | 1M | 3M | YTD | 1Y | Fwd EPS change (%) (b) 1M | 3M | YTD | 1Y | Fwd PER change (%) (c) 1M | 3M | YTD | 1Y | EPS Growth (% y-y) Fwd | 21E | 22E | PER (x) Fwd | 21E | 22E | 4yAvg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSCI | AC World | 742.26 | 0.0 | 0.0 | 1.3 | 4.6 | 2.5 | 2.7 | 14.9 | 28.0 | 0.2 | 4.1 | 24.6 | 32.9 | 2.3 | -1.3 | -7.9 | -3.7 | 12.2 | 49.3 | 7.7 | **18.5** | 19.7 | 18.3 | 16.6 |
| Index | World (DM) | 3,151.16 | 0.0 | 0.1 | 1.3 | 4.8 | 2.5 | 3.4 | 17.1 | 30.2 | 0.6 | 4.7 | 26.0 | 34.1 | 1.9 | -1.2 | -7.1 | -2.9 | 12.4 | 48.2 | 7.9 | **19.5** | 20.9 | 19.3 | 17.3 |
| | N. America | 4,627.53 | -0.2 | 0.1 | 1.6 | 5.8 | 3.5 | 4.1 | 20.8 | 32.4 | 1.0 | 4.7 | 28.8 | 35.6 | 2.5 | -0.6 | -6.2 | -2.3 | 12.3 | 48.0 | 8.4 | **21.5** | 23.1 | 21.4 | 18.7 |

| Code | Name | 연간실적 매출액(십억) 2021E | 2022E | 매출액y-y(%) 2021E | 2022E | 영업이익(십억) 2021E | 2022E | 영업이익y-y(%) 2021E | 2022E | 순이익(십억) 2021E | 2022E | 순이익y-y(%) 2021E | 2022E | 지배순이익(십억) 2021E | 2022E | 지배순이익y-y(%) 2021E | 2022E | 매출액(십억) 1Q21 | 2Q21 | 3Q21 | 4Q21 | 매출액y-y(%) 1Q21 | 2Q21 | 3Q21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A005930 | 삼성전자 | 276,340 | 297,907 | 16.7 | 7.8 | 52,686 | 56,360 | 46.4 | 7.0 | 39,936 | 43,047 | 51.2 | 7.8 | 38,912 | 42,428 | 49.1 | 9.0 | 65,389 | 63,672 | 73,000 | 74,295 | 18.2 | 20.2 | 9.0 |
| A000660 | SK하이닉스 | 42,530 | 49,328 | 33.3 | 16.0 | 12,035 | 13,823 | 140.1 | 14.9 | 9,128 | 10,460 | 91.8 | 14.6 | 9,113 | 10,425 | 91.7 | 14.4 | 8,494 | 10,322 | 11,814 | 11,902 | 18.0 | 19.9 | 45.3 |
| A035420 | NAVER | 6,784 | 8,312 | 27.9 | 22.5 | 1,360 | 1,756 | 11.9 | 29.2 | 15,824 | 1,825 | 1,772.6 | -88.5 | 16,582 | 1,854 | 1,554.8 | -88.8 | 1,499 | 1,664 | 1,727 | 1,900 | -13.5 | -12.6 | 1,000.3 |
| A207940 | 삼성바이오로직스 | 1,491 | 1,690 | 28.0 | 13.4 | 490 | 531 | 67.4 | 8.4 | 396 | 435 | 64.3 | 9.8 | 396 | 435 | 64.2 | 9.8 | 261 | 412 | 411 | 421 | 25.9 | 34.0 | 49.7 |
| A051910 | LG화학 | 44,085 | 49,876 | 46.6 | 13.1 | 5,849 | 4,968 | 225.3 | -15.1 | 4,439 | 3,476 | 550.5 | -21.7 | 4,077 | 3,173 | 695.2 | -22.2 | 9,650 | 11,456 | 11,285 | 11,733 | 35.6 | 74.9 | 50.3 |
| A035720 | 카카오 | 5,987 | 7,607 | 44.0 | 27.1 | 786 | 1,147 | 72.3 | 46.0 | 951 | 1,097 | 448.6 | 15.3 | 902 | 1,029 | 479.4 | 14.1 | 1,258 | 1,352 | 1,644 | 1,733 | 44.9 | 41.9 | 49.4 |
| A006400 | 삼성SDI | 13,871 | 16,666 | 22.8 | 20.2 | 1,170 | 1,577 | 74.3 | 34.8 | 1,194 | 1,498 | 89.2 | 25.5 | 1,119 | 1,410 | 94.6 | 26.0 | 2,963 | 3,334 | 3,617 | 3,957 | 23.6 | 30.3 | 17.2 |
| A005380 | 현대차 | 116,993 | 125,659 | 12.5 | 7.4 | 7,136 | 8,018 | 198.0 | 12.4 | 6,635 | 7,393 | 244.7 | 11.4 | 6,020 | 6,795 | 322.7 | 12.9 | 27,391 | 30,326 | 28,132 | 31,054 | 8.2 | 38.7 | 2.0 |
| A068270 | 셀트리온 | 2,047 | 2,419 | 10.7 | 18.2 | 820 | 963 | 15.2 | 17.4 | 654 | 762 | 26.0 | 16.4 | 641 | 746 | 25.5 | 16.4 | 457 | 432 | 486 | 584 | 22.6 | 0.7 | -11.4 |
| A000270 | 기아 | 70,551 | 75,507 | 19.2 | 7.0 | 5,320 | 5,854 | 157.5 | 10.0 | 4,733 | 5,190 | 218.1 | 9.7 | 4,730 | 5,181 | 217.9 | 9.5 | 16,582 | 18,339 | 17,125 | 18,478 | 13.8 | 61.3 | 4.9 |
| A323410 | 카카오뱅크 | | | | | 296 | 521 | 141.8 | 75.9 | 244 | 399 | 114.3 | 64.0 | 244 | 399 | 114.3 | 64.0 | 225 | 254 | | | | | |
| A005490 | POSCO | 72,925 | 74,829 | 26.2 | 2.6 | 8,884 | 8,158 | 269.7 | -8.2 | 6,603 | 5,877 | 269.3 | -11.0 | 6,094 | 5,425 | 280.4 | -11.0 | 16,069 | 18,292 | 20,610 | 19,280 | 10.5 | 33.3 | 44.5 |
| A012330 | 현대모비스 | 41,047 | 45,462 | 12.1 | 10.8 | 2,228 | 2,706 | 21.8 | 21.4 | 2,616 | 3,102 | 71.3 | 18.6 | 2,599 | 3,081 | 70.0 | 18.5 | 9,816 | 10,285 | 9,900 | 11,004 | 16.5 | 36.5 | -0.9 |
| A259960 | 크래프톤 | 2,128 | 3,306 | 27.4 | 55.4 | 907 | 1,531 | 17.2 | 68.9 | 722 | 1,197 | 29.8 | 65.9 | 722 | 1,197 | 29.8 | 65.9 | 461 | 459 | 460 | 757 | -9.3 | 21.2 | 31.4 |
| A028260 | 삼성물산 | 32,740 | 32,926 | 8.4 | 0.6 | 1,407 | 1,448 | 64.2 | 2.9 | 1,923 | 1,796 | 65.7 | -6.6 | 1,730 | 1,563 | 67.1 | -9.6 | 7,840 | 8,548 | 8,034 | 8,377 | 12.6 | 18.3 | 2.3 |
| A066570 | LG전자 | 71,956 | 74,202 | 13.7 | 3.1 | 4,288 | 4,993 | 34.2 | 16.4 | 2,109 | 3,675 | 2.2 | 74.3 | 1,821 | 3,615 | -7.5 | 98.5 | 18,809 | 16,117 | 18,785 | 17,986 | 27.7 | 25.6 | 11.0 |
| A096770 | SK이노베이션 | 45,612 | 52,196 | 33.5 | 14.4 | 1,938 | 2,132 | 175.5 | 10.0 | 901 | 1,430 | 142.0 | 58.8 | 878 | 1,401 | 140.4 | 59.5 | 9,240 | 11,120 | 12,270 | 13,120 | -17.2 | 54.5 | 45.7 |
| A017670 | SK텔레콤 | 19,538 | 20,303 | 4.9 | 3.9 | 1,526 | 1,678 | 13.1 | 10.0 | 2,655 | 2,905 | 77.0 | 9.4 | 2,634 | 2,834 | 75.1 | 7.6 | 4,780 | 4,818 | 4,914 | 5,036 | 7.4 | 4.7 | 3.9 |
| A105560 | KB금융 | | | | | 6,172 | 6,351 | 33.7 | 2.9 | 4,497 | 4,688 | 28.4 | 4.2 | 4,463 | 4,646 | 29.2 | 4.1 | 17,584 | 12,242 | 16,473 | | -8.8 | 14.2 | 56.4 |
| A051900 | LG생활건강 | 8,401 | 9,115 | 7.1 | 8.5 | 1,313 | 1,463 | 7.5 | 11.5 | 892 | 1,005 | 9.7 | 12.7 | 874 | 984 | 9.6 | 12.6 | 2,037 | 2,021 | 2,139 | 2,203 | 7.4 | 13.4 | 3.3 |
| A055550 | 신한지주 | | | | | 5,993 | 6,188 | 21.6 | 3.3 | 4,334 | 4,484 | 23.9 | 3.5 | 4,292 | 4,414 | 25.7 | 2.8 | 6,895 | 6,813 | | | -15.3 | 11.6 | |
| A302440 | SK바이오사이언스 | 976 | 3,572 | 332.4 | 266.2 | 438 | 1,578 | 1,060.7 | 260.5 | 355 | 1,227 | 980.7 | 245.2 | 355 | 1,227 | 980.4 | 245.3 | 113 | 145 | 173 | 499 | 397.4 | 266.8 | 78.8 |
| A034730 | SK | 94,268 | 100,607 | 15.2 | 6.7 | 4,642 | 5,335 | 2,922.4 | 14.9 | 3,106 | 3,609 | 2,964.8 | 16.2 | 1,377 | 1,642 | 627.2 | 19.3 | 22,250 | 23,406 | 23,724 | 25,547 | -6.2 | 24.7 | 18.9 |
| A091990 | 셀트리온헬스케어 | 1,838 | 2,179 | 12.9 | 18.5 | 331 | 438 | -8.6 | 32.5 | 257 | 336 | 7.0 | 30.8 | 257 | 336 | 7.0 | 30.8 | 356 | 433 | 488 | 569 | -0.2 | 3.1 | 5.2 |
| A361610 | SK아이이테크놀로 | 676 | 1,076 | 44.1 | 59.2 | 189 | 333 | 50.7 | 76.3 | 148 | 255 | 68.1 | 72.0 | 149 | 246 | 69.3 | 65.1 | 139 | 155 | 178 | 227 | | | |
| A015760 | 한국전력 | 59,706 | 62,797 | 1.9 | 5.2 | -1,631 | -951 | -139.9 | 41.7 | -2,165 | -1,710 | -203.5 | 21.0 | -2,230 | -1,755 | -212.0 | 21.3 | 15,075 | 13,519 | 16,039 | 15,091 | -0.1 | 3.4 | 2.1 |
| A011200 | HMM | 12,561 | 12,447 | 95.9 | -0.9 | 6,315 | 5,669 | 543.9 | -10.2 | 3,837 | 4,905 | 2,995.0 | 27.8 | 3,866 | 4,994 | 3,020.8 | 29.2 | 2,428 | 2,907 | 3,592 | 3,489 | 84.9 | 111.4 | 109.0 |
| A003550 | LG | 8,303 | 8,612 | 25.2 | 3.7 | 2,797 | 2,957 | 64.3 | 5.7 | 3,054 | 2,606 | 98.3 | -14.7 | 2,958 | 2,507 | 101.8 | -15.2 | 2,180 | 1,924 | 2,011 | 2,214 | 37.5 | 28.7 | 2.8 |
| A032830 | 삼성생명 | 20,270 | 20,905 | -41.3 | 3.1 | 1,182 | 689 | -34.0 | -41.7 | 1,755 | 1,399 | 28.1 | -20.3 | 1,653 | 1,359 | 30.6 | -17.8 | 10,008 | 8,195 | 4,701 | 4,759 | -3.5 | 7.3 | -39.5 |
| A086790 | 하나금융지주 | | | | | 4,393 | 4,649 | 14.5 | 5.8 | 3,259 | 3,416 | 21.4 | 4.8 | 3,200 | 3,357 | 21.4 | 4.9 | 14,011 | 6,908 | 13,195 | | -32.0 | 10.8 | 102.5 |

## Compliance notice

- 보고서는 철저히 계량적 분석에 근거한 의견을 제시합니다. 따라서 당사의 대표 투자의견과 다를 수 있습니다.
- 본 조사분석자료의 애널리스트는 2026년 1월 20일 현재 위 조사분석자료에 언급된 종목의 지분을 보유하고 있지 않습니다.
- 당사는 2026년 1월 20일 현재 위 조사분석자료에 언급된 종목의 지분을 1% 이상 보유하고 있지 않습니다.
- 본 조사분석자료에는 외부의 부당한 압력이나 간섭 없이 애널리스트의 의견이 정확하게 반영되었음을 확인합니다.
- 본 조사분석자료는 당사의 저작물로서 모든 저작권은 당사에게 있습니다.
- 본 조사분석자료는 당사의 동의 없이 어떠한 경우에도 어떠한 형태로든 복제, 배포, 전송, 변형, 대여할 수 없습니다.
- 본 조사분석자료에 수록된 내용은 당사 리서치센터가 신뢰할 만한 자료 및 정보로부터 얻어진 것이나, 당사는 그 정확성이나 완전성을 보장할 수 없습니다. 따라서 어떠한 경우에도 본 자료는 고객의 주식투자의 결과에 대한 법적 책임소재에 대한 증빙자료로 사용될 수 없습니다.
- 본 조사분석자료는 기관투자가 등 제3자에게 사전 제공된 사실이 없습니다.

# 삼성증권

**삼성증권주식회사**
서울특별시 서초구 서초대로74길 11(삼성전자빌딩)
Tel: 02 2020 8000 / www.samsungpop.com

**삼성증권 Family Center:** 1588 2323

**고객 불편사항 접수:** 080 911 0900

Member of
Dow Jones
Sustainability Indices
Powered by the S&P Global CSA