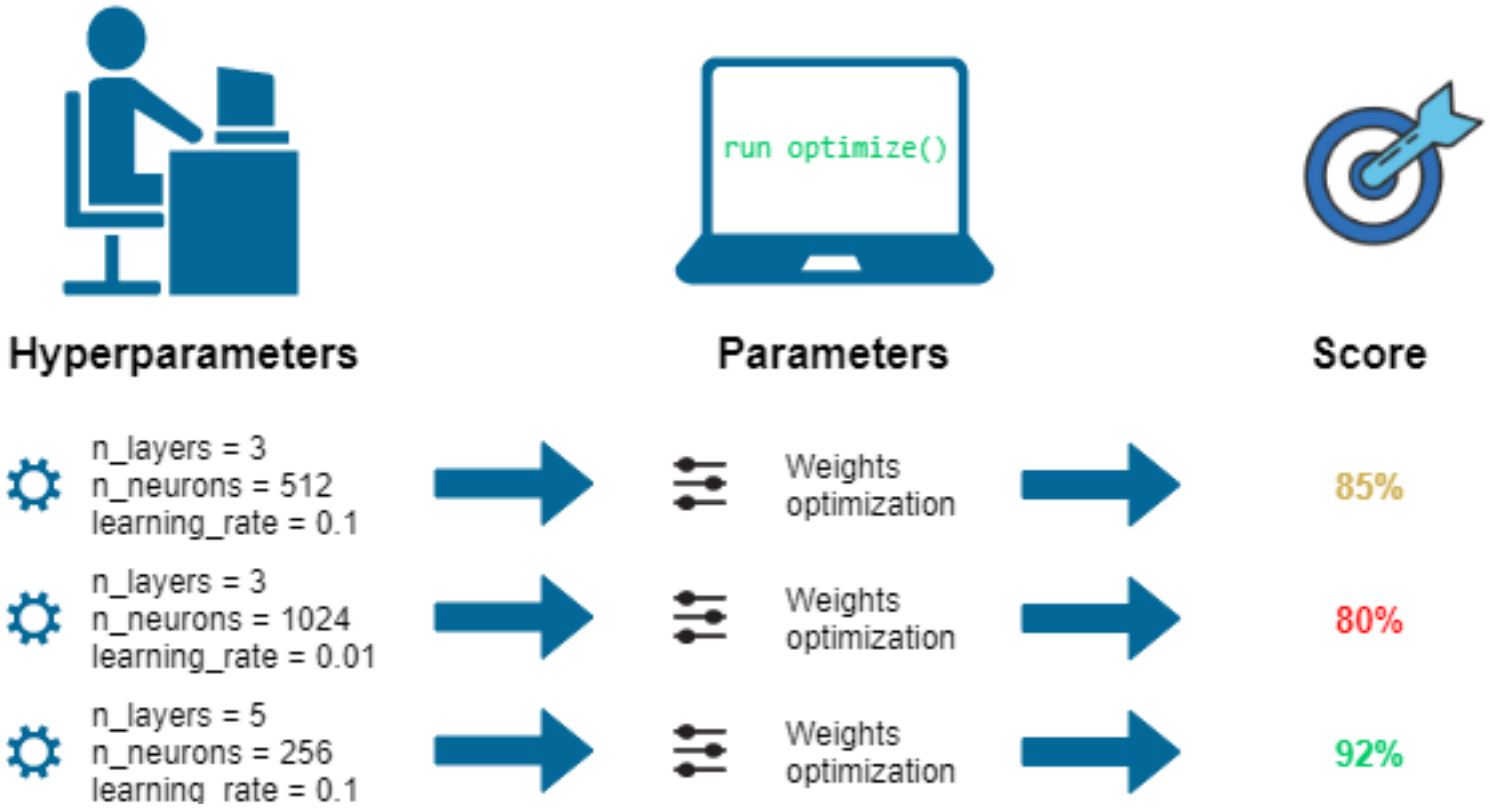
반도체장비예지보전실무 06

시계열 데이터 기반 AI 개발

김민진 | muea.mj@gmail.com

하이퍼파라미터 튜닝

모델 파라미터 vs. 하이퍼파라미터 개념



https://www.kdnuggets.com/2020/02/practical-hyperparameter-optimization.html

모델 파라미터 vs. 하이퍼파라미터 개념

- 모델 파라미터(model parameter)
  - 머신러닝 알고리즘이 학습을 통해 찾은 값
  - 머신러닝 알고리즘의 학습 과정은 최적의 모델 파라미터를 찾는 과정과 같음

- 하이퍼파라미터(hyperparameter)
  - 학습하는 값이 아니라 사전에 사람이 지정해야 하는 값
  - 사이킷런에서 하이퍼파라미터는 클래스와 메서드의 매개변수로 표현
  - 모델 검증 과정은 최적의 하이퍼파라미터를 찾는 과정과 같음

하이퍼파라미터 튜닝 실습 (1)

- CNN의 개수, Unit 개수, Dropout의 위치와 비율을 변화시켜가며 결과를 확인해보세요.
  - Unit의 개수는 정해진 게 없음.
  - 이전과 동일(32 → 32), 이전보다 크게(32 → 64), 이전보다 작게(64 → 32)

```
model = Sequential()
model.add(Input(shape=(64, 64, 3)))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

하이퍼파라미터 튜닝 실습 (2)

- Epochs 개수를 변화시켜가며 결과를 관찰하세요.

```
history = model.fit(train_generator,
          epochs=10,
          steps_per_epoch=len(train_generator),
          validation_data=valid_generator,
          validation_steps=len(valid_generator))
```

- 학습률(learning_rate)을 다르게 하여 학습한 후 결과를 관찰하세요.

```
rmsprop = keras.optimizers.RMSprop(learning_rate=1e-2)
dnn_model.compile(optimizer=rmsprop, loss='binary_crossentropy', metrics=['accuracy'])
```

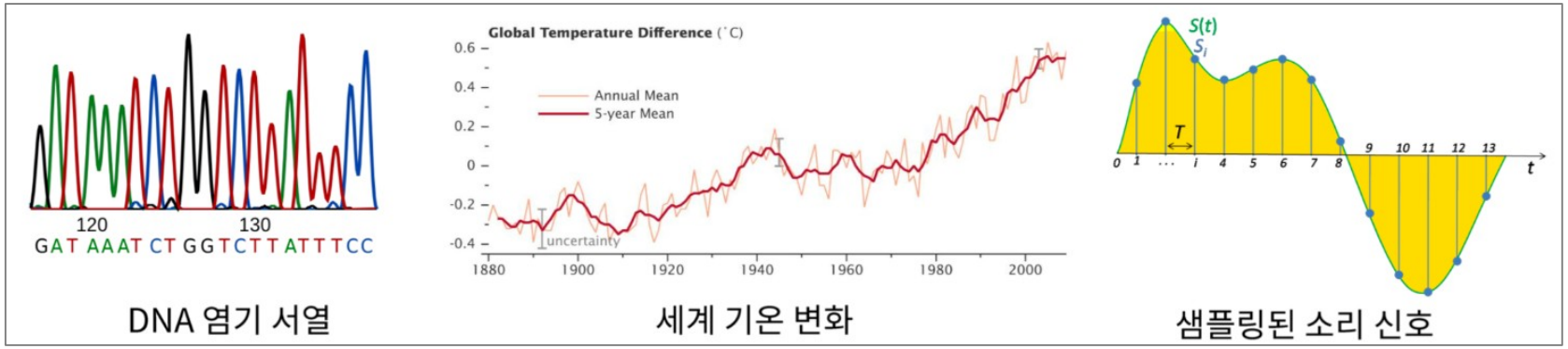
- 모델 학습의 결과에 영향을 미치는 요소는 무엇인가요? 정리해보세요.

시계열 데이터

순차 데이터

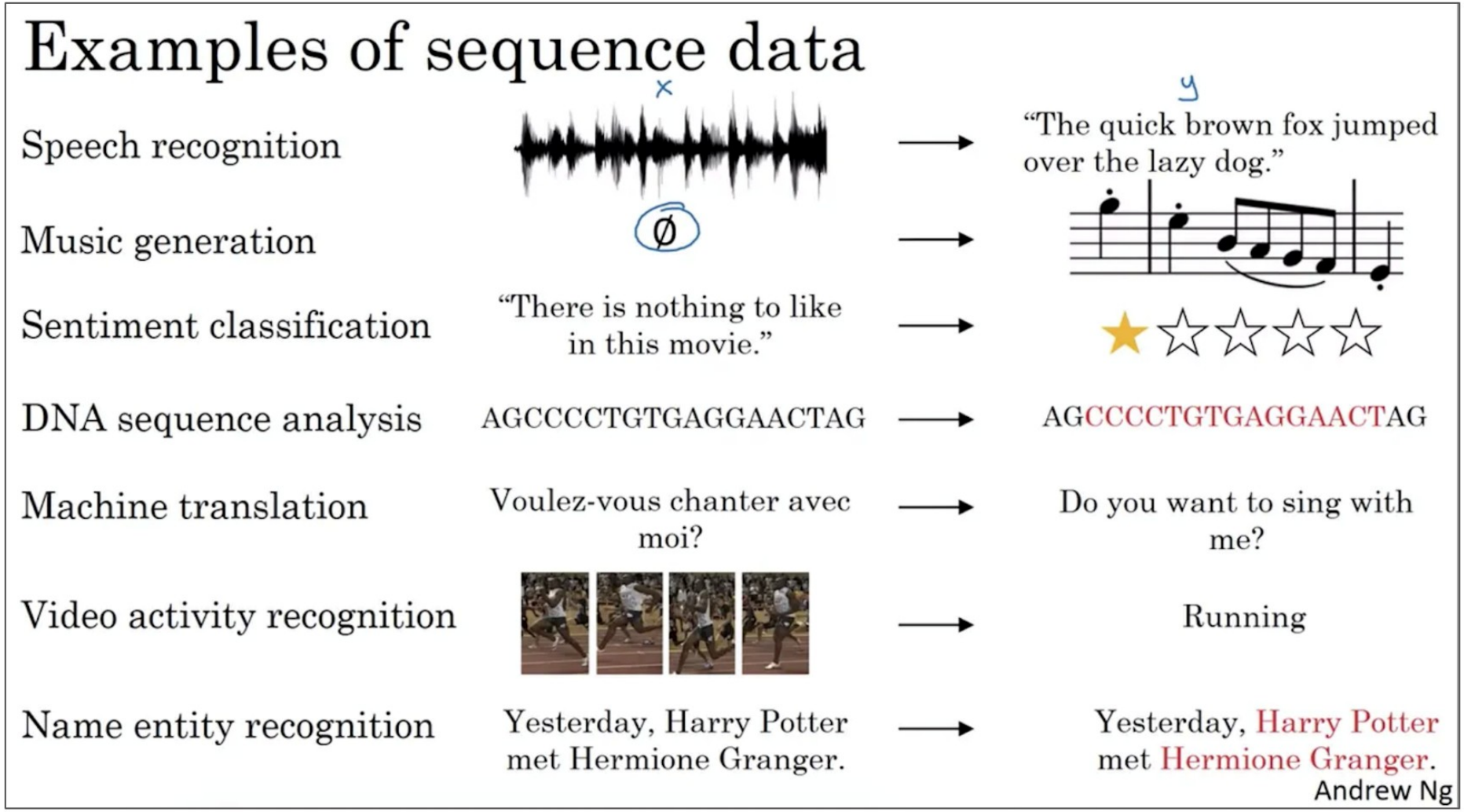
- 순차 데이터(Sequential Data) 개념
  - 시간 순서나 연속적인 관계를 가지고 배열된 데이터
  - 데이터의 순서가 매우 중요하며, 순서가 바뀌면 정보의 의미가 크게 달라지거나 완전히 사라질 수 있음

- 순차 데이터의 특징
  - 연속성 : 데이터가 끊어지지 않고 이어지는 성질을 가지고 있음
  - 길이 가변성 : 순차 데이터는 그 길이가 고정되어 있지 않고 가변적일 수 있음.



https://heung-bae-lee.github.io/2020/01/12/deep_learning_08/

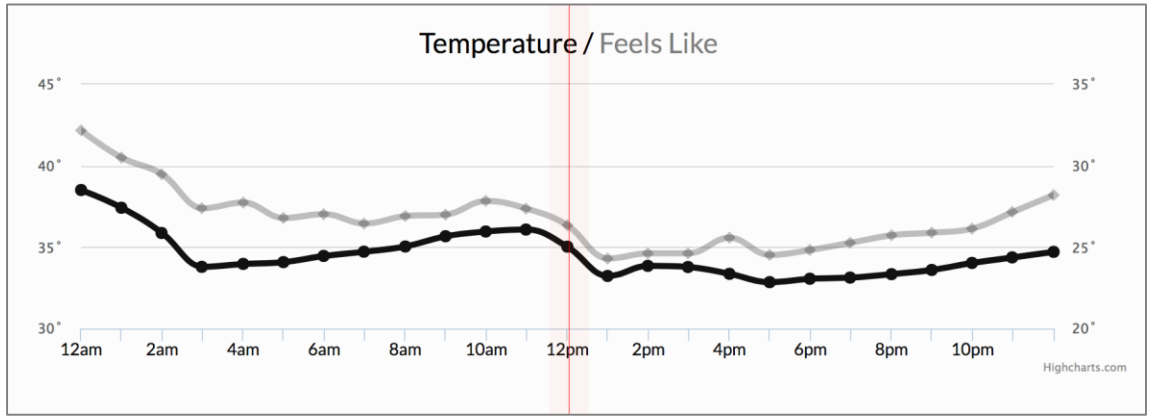
순차 데이터



https://aiml.com/what-does-sequential-data-mean-which-models-are-best-suited-for-handling-sequential-data/

시계열 데이터(Time Series Data)

- **개념 :** 일정한 시간 간격을 두고 시간 순서대로 기록된 데이터 점들의 집합
- **분야별 시계열 데이터 예시**
  - **금융:** 매일 기록되는 주가지수, 환율 변동, 분 단위 거래량
  - **기상 및 환경:** 매시간 측정되는 기온, 습도, 강수량, 미세먼지 농도
  - **유통 및 비즈니스:** 월간 제품 판매량, 일일 웹사이트 방문자 수, 분기별 매출액
  - **제조 및 IoT:** 공장 설비의 센서가 1초마다 기록하는 온도·압력 데이터, 전력 사용량
  - **의료:** 실시간 환자의 심박수, 혈압 모니터링 기록



https://www.influxdata.com/what-is-time-series-data/

시계열 데이터(Time Series Data)

- 주요 특징

시간 의존성	- 데이터 포인트들이 시간 순서에 따라 기록되므로, 현재 값은 과거의 값에 영향을 받음 - 어제의 주가 데이터는 오늘의 주가에 영향을 미칠 수 있지만, 내일의 주가가 오늘의 주가에 영향을 줄 수는 없음
연속성	- 시간의 흐름에 따라 연속적으로 관측됨 - 매분, 매초, 매일 등 일정한 간격으로 기록되는 경우가 많으며, 타임스탬프가 중요
결측치 및 이상치	- 센서 오류나 외부 요인으로 인해 데이터가 누락되거나 비정상적인 값이 나타날 수 있음
다양한 노이즈	- 분석 결과에 영향을 미칠 수 있는 다양한 종류의 노이즈를 포함할 수 있음

시계열 데이터의 구성요소

- 시계열 데이터는 보통 다음 4가지 패턴이 결합되어 나타남.
  - 추세(Trend) : 시간의 흐름에 따라 장기적인 증가 또는 감소 경향
  - 계절성(Seasonality) : 특정 주기마다 반복적으로 나타나는 패턴
  - 주기성(Cycle) : 일정하지 않은 주기를 가지며 경제적/사회적 요인에 의해 발생하는 장기적 변화
  - 불규칙성(Irregular) : 위 요소들로 설명되지 않는 임의의 변동

시계열 데이터의 구성요소

Topic	추세 (Trend)	계절성 (Seasonality)	주기성 (Cyclic)	불규칙성 (Irregularity)
일정 시간 간격 (Fixed Time Interval)	Yes	Yes	No	No
기간 유형 (Movement Type)	Long/Short Term	Short term	Long/Short Term	Random/Irregular
그래프 (Grapy)	Positive Trend	Seasonality	Cyclic	Irregular
예시 (Example))	시간이 지남에 따른 판매 증가	여름마다 아이스크림 판매 증가	경제 성장률 변동	시간에 따른 주식 가격

https://wikidocs.net/293085

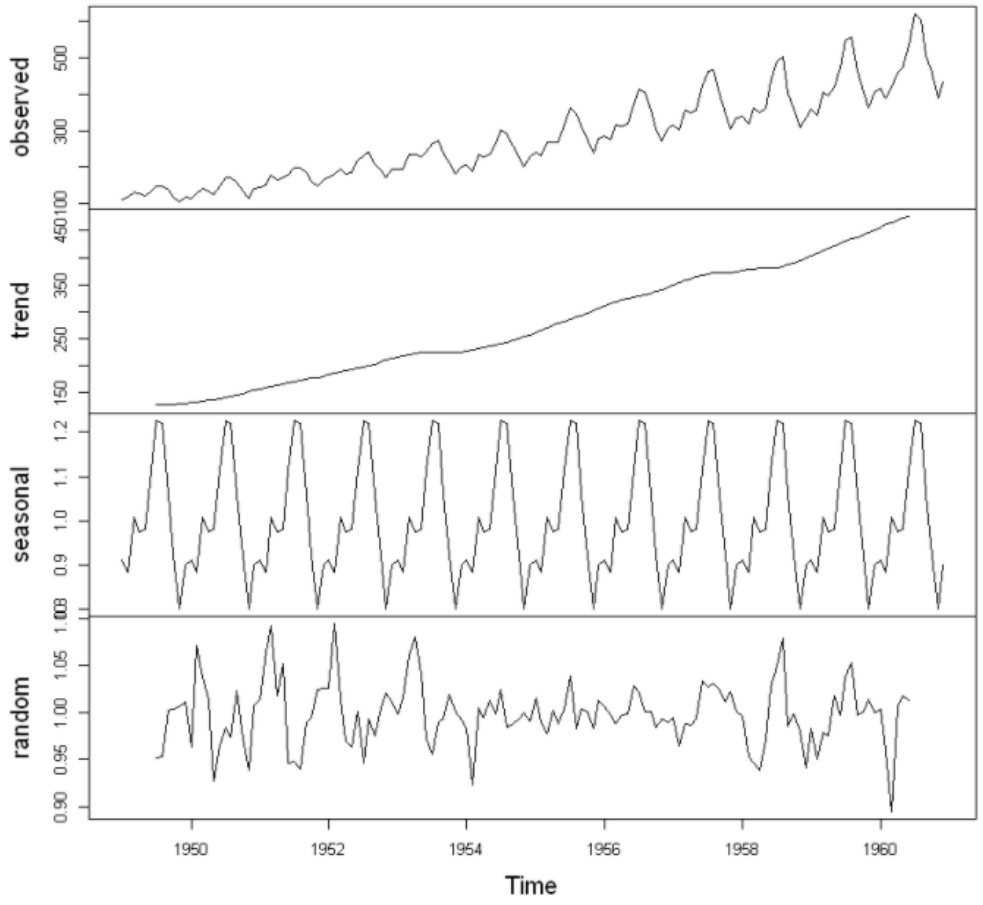
시계열 데이터 읽기

- Observed (전체 데이터)
  - 맨 위의 그래프는 관찰된 전체 시계열 데이터, 아래의 그래프에서 각각의 요소(추세, 계절성, 잔차)로 분해

- Trend (추세, T)
  - 이 그래프에서 볼 수 있듯이, 추세는 점진적으로 상승하고 있으며, 이는 전반적인 증가 경향을 시사

- Seasonal (계절성, S)
  - 이 그래프에서는 일정한 주기를 가지고 반복되는 변동 패턴을 확인

- Residual (잔차, R)
  - 이 그래프에서는 데이터의 예측 오차 또는 불규칙한 변동성을 볼 수 있음.



시계열 데이터 분석의 목표

- 미래 예측 (Forecasting)
  - 과거 데이터의 패턴을 분석하여 미래의 값을 예측
  - 예) 내일의 주가나 다음 분기의 제품 판매량을 예측

- 이상 탐지 (Anomaly Detection)
  - 정상적인 데이터 패턴에서 벗어나는 특이한 값(이상치)을 찾아냄
  - 예) 공장 설비의 센서 데이터에서 이상 신호를 감지하거나 신용카드 부정 사용을 탐지

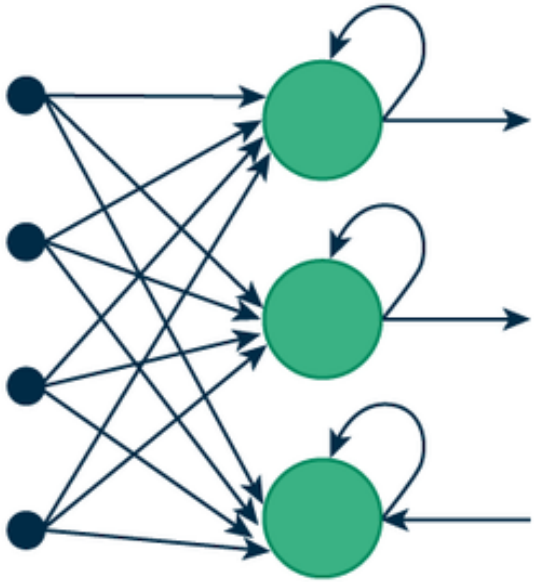
- 패턴 및 경향 분석
  - 데이터에 내재된 추세, 계절성 등의 패턴을 파악하여 비즈니스 인사이트를 얻음

순환 신경망

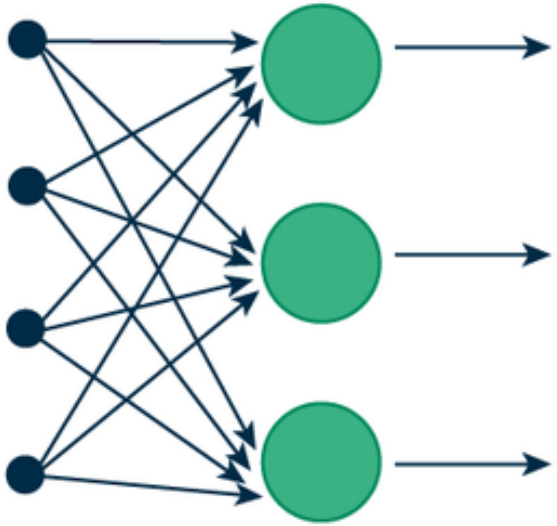
(RNN: Recurrent Neural Network)

순환 신경망 개념

- 순환 신경망 (RNN: Recurrent Neural Network)
  - 시계열 데이터와 같이 순차적인 입력을 처리하고, 과거의 정보를 기억하여 현재를 처리하는 딥러닝 모델



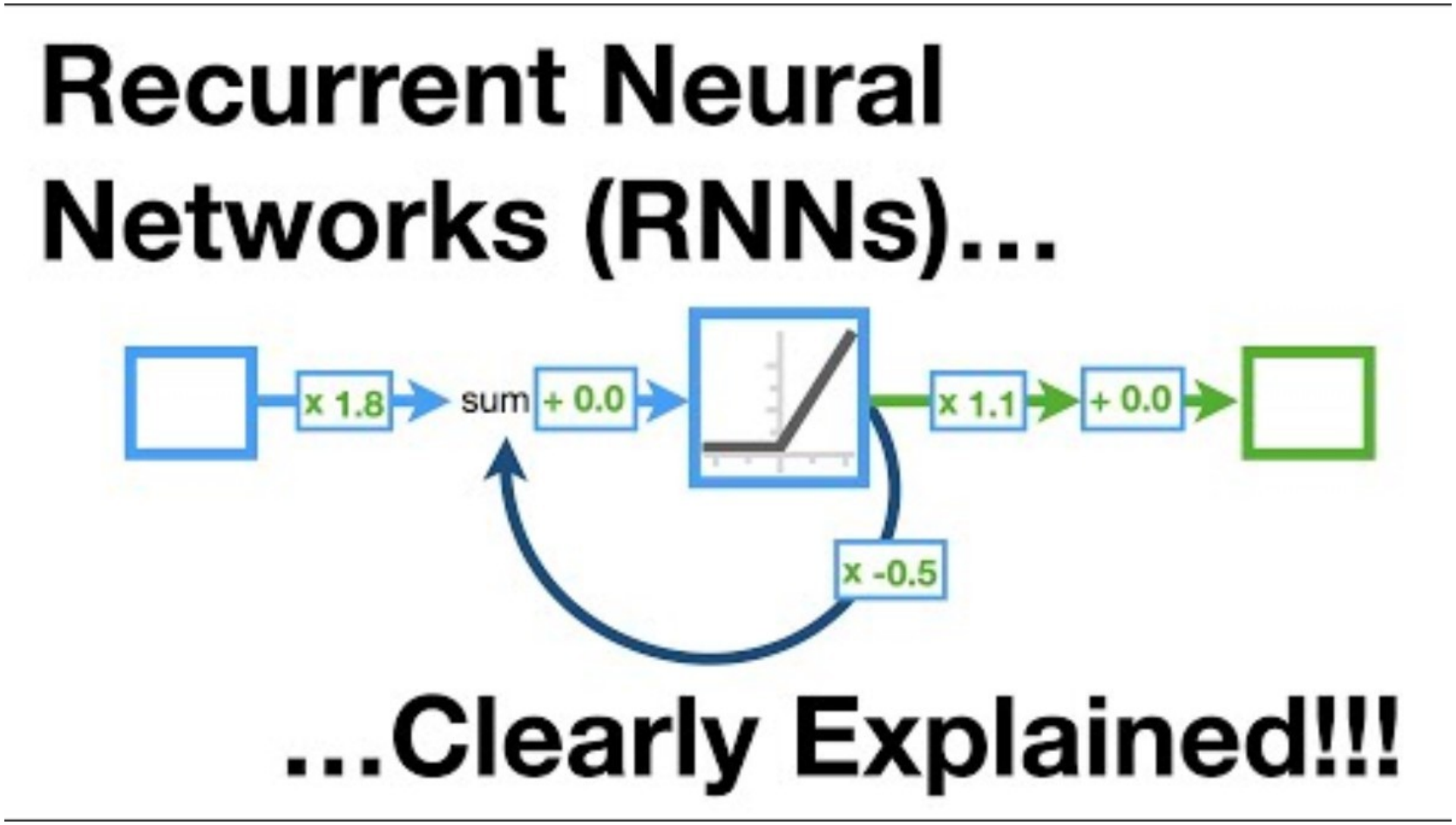
(a) Recurrent Neural Network



(b) Feed-Forward Neural Network

https://www.geeksforgeeks.org/machine-learning/introduction-to-recurrent-neural-network/

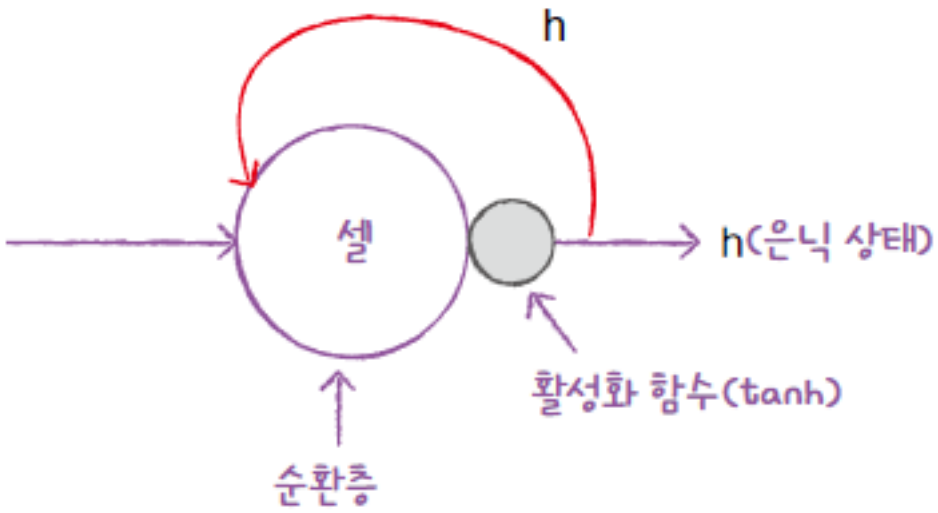
순환 신경망 개념



https://youtu.be/AsNTP8Kwu80?si=pxkrxDV4LXJ6KWwR&t=114

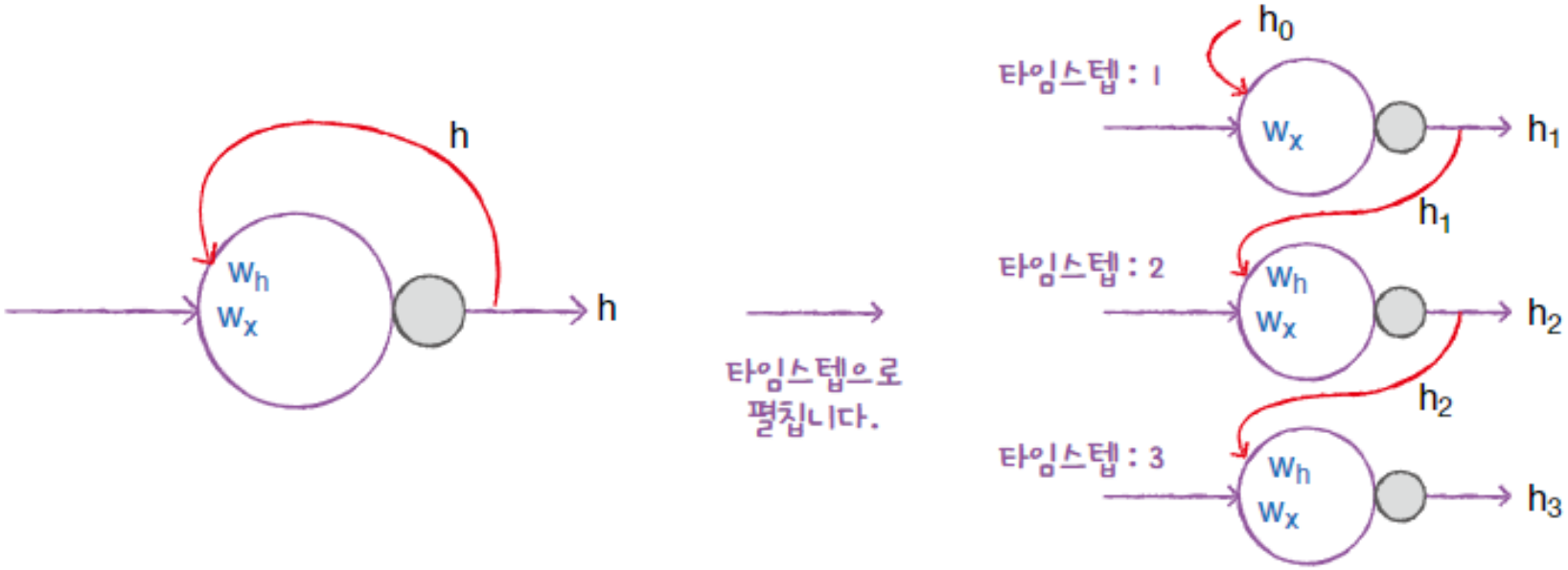
순환 신경망 : 동작 원리

- 셀(cell) : 순환층/순환 레이어
- 은닉상태(hidden state) : 이전 단계에서 계산된 정보가 저장된 상태



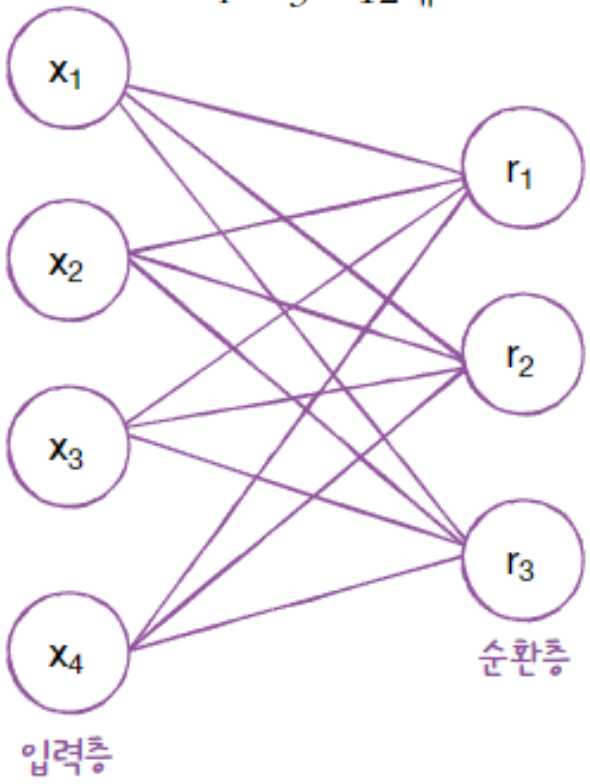
순환 신경망 : 동작 원리

- 타임스텝(timestep) : 시퀀스를 처리하는 한 단계

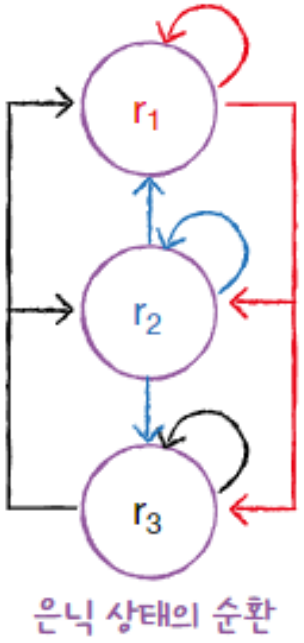


순환 신경망 : 동작 원리

$4 \times 3 = 12$개



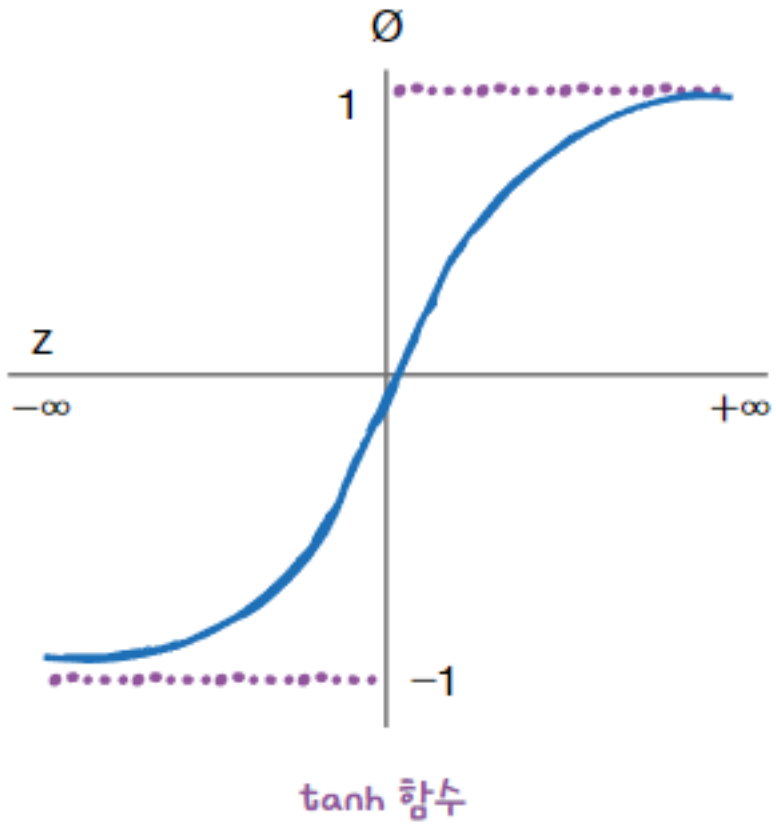
$3 \times 3 = 9$개



$$\text{모델 파라미터 수} = w_x + w_b + \text{절편} = 12 + 9 + 3 = 24$$

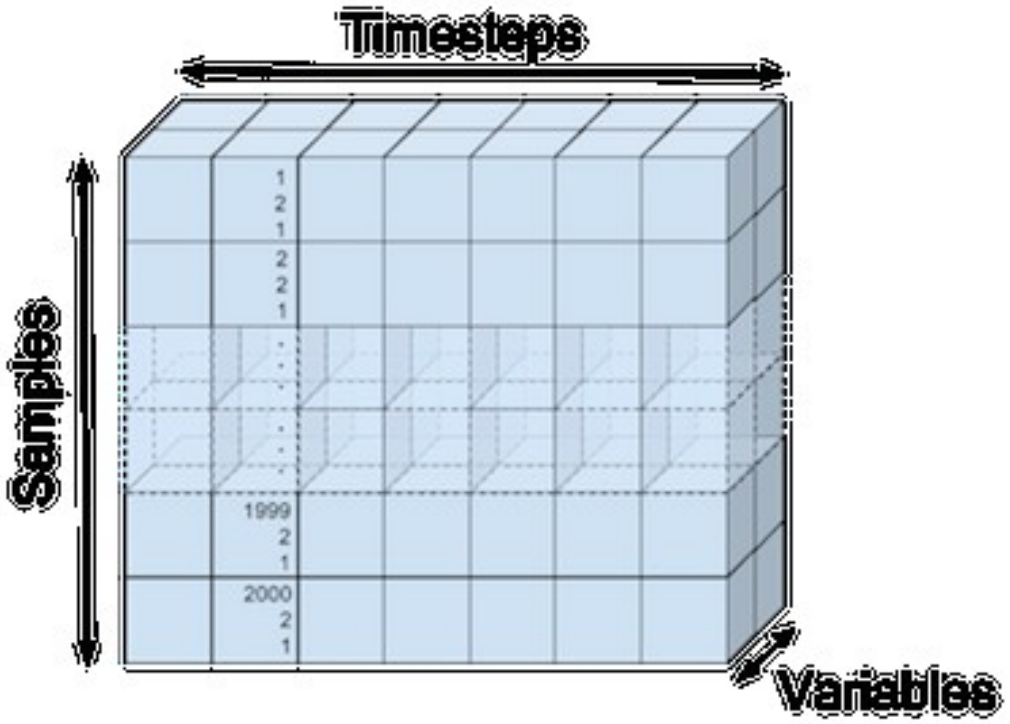
순환 신경망 : 활성화 함수

- 하이퍼볼릭 탄젠트(hyperbolic tangent) 함수



순환 신경망의 입력

- 입력의 형태 : *(Samples, Timesteps, Variables)*
  - Timesteps (시간 단계)
    - 시간의 흐름에 따라 관측된 순서.
    - 1분마다 측정했다면 t1, t2, t3, …처럼 이어지는 각 시점에 해당
  - Variables (변수/특성)
    - 각 시점에서 함께 측정하는 데이터 항목.
    - 온도, 압력, 진동처럼 여러 센서값이 될 수 있다.
  - Samples (샘플)
    - 하나의 시계열 묶음이 몇 개 있는지를 의미
  - Sequence (시퀀스)
    - 여러 개의 timestep이 시간 순서대로 이어진 한 묶음



- 예시: **장비 100대에서 10분 동안 온도·압력·진동을 측정**했다면 입력 데이터의 크기는 (100, 10, 3)

순환 신경망의 입력 : Sliding Window

- Sliding Window
  - 개념
    - 긴 시계열 데이터를 일정한 길이의 구간(Window)으로 나누어 여러 개의 학습 데이터로 만드는 방법
  - 요소
    - Window Size: 한 번에 사용할 timestep의 개수
    - Stride: Window를 다음 위치로 이동시키는 간격
  - 결과물
    - 잘라낸 각 Window가 하나의 Sequence이자 하나의 Sample이 됨

순환 신경망의 입력 : Sliding Window

- Sliding Window 예시

원본 시계열 데이터

[1, 2, 3, 4, 5, 6, 7]

Window Size = 3, Stride = 1

```
Sample 1 : [1, 2, 3]
Sample 2 :    [2, 3, 4]
Sample 3 :       [3, 4, 5]
Sample 4 :          [4, 5, 6]
Sample 5 :             [5, 6, 7]
```

→하나의 긴 시계열 데이터에서 **5개의 Sample 생성**

→Sample 수

= (전체 timestep 수 – Window Size) / Stride + 1

※ 나누어떨어지지 않는 경우에는 정수 부분만 사용

확인문제

1. 다음 중 순차 데이터로 처리하기 어려운 작업은 무엇인가요?

① 환자의 검사 결과를 바탕으로 질병 예측하기

② 월별 주택 가격을 바탕으로 다음 달의 주택 가격 예측하기

③ 태풍의 이동 경로를 바탕으로 다음 경로 예측하기

④ 노래 악보를 바탕으로 다음 음표를 예측하기

2. 순환 신경망에서 순환층을 부르는 다른 말과 순환층의 출력을 나타내는 용어를 올바르게 짝지은 것은 무엇인가요?

① 셀(shell)–셀 상태

② 셀(shell)–은닉 상태

③ 셀(cell)–셀 상태

④ 셀(cell)–은닉 상태

확인문제

3. 순환 신경망에서 한 셀에 있는 뉴런의 개수가 10개입니다. 이 셀의 은닉 상태가 다음 타임스텝에 사용될 때 곱해지는 가중치 w_h의 크기는 얼마인가요?

① (10,)

② (10, 10)

③ (10, 10, 10)

④ 알 수 없음

실습1. 사인(Sine) 함수 예측 모델 만들기

- **실습 내용:** 과거 10개의 사인 값을 보고, 11번째 값을 예측하는 모델을 만듭니다.

- **실습 단계:**
  ① 데이터 준비
  ② RNN에 입력할 수 있도록 전처리
  ③ 모델링 및 학습
  ④ 결과 확인

① 데이터 준비

- numpy를 이용해 아름다운 곡선을 가진 사인 함수 데이터를 생성합니다.

```
import numpy as np

# 0부터 50까지 0.1 간격으로 숫자를 생성하여 사인 값 계산
time_steps = np.arange(0, 50, 0.1)
data = np.sin(time_steps)
```

② RNN에 입력할 수 있도록 전처리

- RNN이 학습할 수 있도록, 데이터를 '문제지(X)'와 '정답지(y)' 형태로 만들어 줍니다.
- Sliding Window : 창문을 한 칸씩 옮겨가 듯 데이터를 자릅니다.
  - 문제 1: [0~9번째 값] -> 정답 1: [10번째 값]
  - 문제 2: [1~10번째 값] -> 정답 2: [11번째 값]

```
sequence_length = 10 # 과거 10개의 데이터를 보고 다음을 예측
X = []
y = []

for i in range(len(data) - sequence_length):
    X.append(data[i : i + sequence_length])
    y.append(data[i + sequence_length])

X = np.array(X)
y = np.array(y)
```

② RNN에 입력할 수 있도록 전처리

- Keras의 RNN 레이어는 3차원 데이터를 입력으로 받음 → **[샘플 수, 타임스텝, 특성 수]**
- **Reshape 필요**
  - 샘플 수: 전체 문제지의 개수
  - 타임스텝 (timesteps): 한 문제당 포함된 데이터의 길이 (여기서는 sequence_length인 10)
  - 특성 수 (features): 각 시점의 데이터가 몇 개의 값으로 이루어져 있는지 (사인 값 1개이므로 1)

```
# [490, 10] 형태의 데이터를 [490, 10, 1] 형태로 변경
X = X.reshape(X.shape[0], X.shape[1], 1)
print(f"X 데이터 최종 형태: {X.shape}") # (490, 10, 1)
```

③ 모델링 및 학습

- TensorFlow와 Keras를 이용해 **SimpleRNN 모델**을 만듭니다.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

model = Sequential([
    # input_shape는 첫 번째 레이어에만 지정
    SimpleRNN(units=32, input_shape=(10, 1)),
    Dense(units=1) # 다음 숫자 1개를 예측하므로 units=1
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X, y, epochs=20, batch_size=16)
```

③ 모델링 및 학습

- 케라스API: SimpleRNN
  - 순환 신경망의 가장 기본적인 형태를 구현하는 클래스
  - 과거의 정보를 기억하는 '은닉 상태(hidden state)'를 사용하여 시퀀스 데이터를 처리

units	• 순환 신경망의 은닉 상태(Hidden State)를 구성하는 **뉴런의 개수** – 이 값이 클수록 모델은 더 많은 정보를 기억 – 하지만 지나치게 큰 값은 과적합을 유발하거나, 학습 시간을 늘릴 수 있음
activation	• 활성화 함수를 지정 –'tanh': 기본값으로 많이 사용되며, 출력값을 –1과 1 사이로 정규화

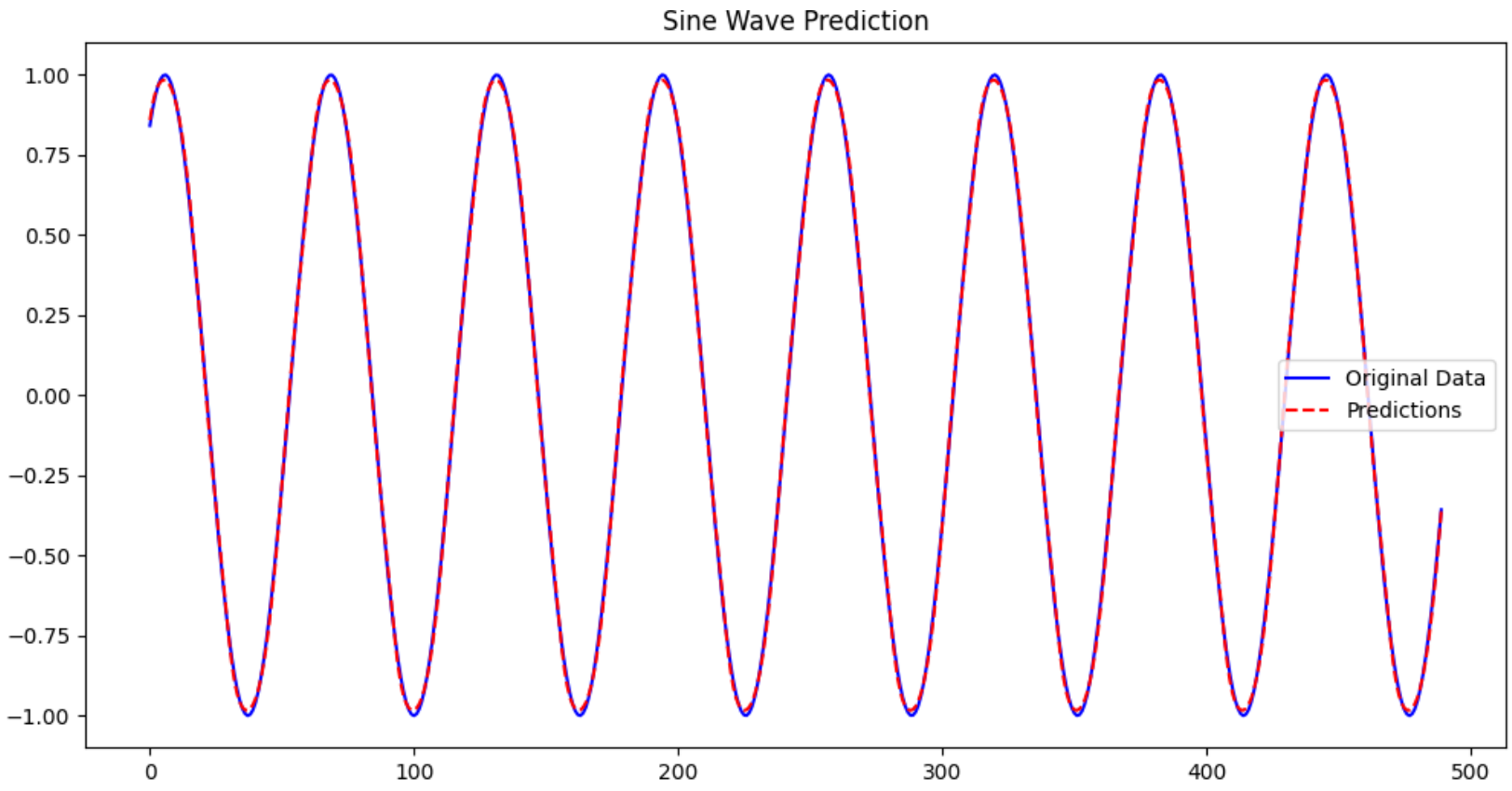
④ 결과 확인

- 학습된 모델이 얼마나 예측을 잘하는지, 실제 사인파와 예측된 값을 함께 그려서 확인해 봅니다.

```
import matplotlib.pyplot as plt

predictions = model.predict(X)

plt.figure(figsize=(12, 6))
# 정답 데이터(y)는 파란색 실선으로 표시
plt.plot(y, color='blue', label='Original Data')
# 모델 예측(predictions)은 빨간색 점선으로 표시
plt.plot(predictions, color='red', linestyle='--', label='Predictions')
plt.legend()
plt.title('Sine Wave Prediction')
plt.show()
```

④ 결과 확인

- 실행결과


Sine Wave Prediction
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
100
200
300
400
500
Original Data
Predictions

실습2-1. Apple 주가 예측하기 : SimpleRNN

- **실습 내용:** Apple의 과거 주가 데이터를 학습하여 다음 날 주가를 예측하는 모델 만들기

- **실습 단계:**

  ① 데이터 준비 및 정규화

  ② 데이터 전처리

  ③ RNN 모델 학습용 데이터셋 구축

  ④ SimpleRNN 모델 학습하기

  ⑤ 학습 결과 확인

① 데이터 준비 및 정규화

- yfinance로 Apple의 과거 주가 데이터를 불러온다.
- MinMaxScaler를 사용해 모든 특성의 범위를 0과 1 사이로 정규화한다.

```python
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler

# Apple(AAPL) 주가 데이터 불러오기
df = yf.download('AAPL', start='2015-01-01')

# 사용할 특성만 선택 (예: 시가, 고가, 저가, 종가, 거래량)
df_features = df[['Open', 'High', 'Low', 'Close', 'Volume']]

# 데이터 확인
df_features.info()
print("="*50)
print(df_features.head())

# 데이터 정규화
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df_features)
```

```
DatetimeIndex: 2953 entries, 2015-01-02 to 2026-09-30
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   (Open, AAPL)    2953 non-null   float64
 1   (High, AAPL)    2953 non-null   float64
 2   (Low, AAPL)     2953 non-null   float64
 3   (Close, AAPL)   2953 non-null   float64
 4   (Volume, AAPL)  2953 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 138.4 KB
==================================================
Price            Open       High        Low      Close     Volume
Ticker           AAPL       AAPL       AAPL       AAPL       AAPL
Date
2015-01-02  24.627203  24.638258  23.734000  24.171759  212818400
2015-01-05  23.941824  24.021417  23.305086  23.490801  257142000
2015-01-06  23.554918  23.751688  23.132636  23.493013  263188400
2015-01-07  23.700837  23.921927  23.590292  23.822437  160423600
2015-01-08  24.149645  24.795227  24.032466  24.737743  237458000
```

② 데이터 전처리

- Pandas의 **rolling 함수**로 **새로운 특성**들을 만든다.

- 결측치를 제거한다.

```python
import pandas as pd

# 시퀀스 길이(타임스텝) 설정
# 과거 60일치의 데이터를 보고 다음 날을 예측
seq_len = 60

# 정규화된 데이터를 다시 DataFrame으로 만들어 특성 공학 적용
df_scaled = pd.DataFrame(scaled_data,
                         columns=df_features.columns,
                         index=df_features.index)

# 1. 이동 평균 (Moving Average) - 추세 파악
df_scaled['MA_20'] = df_scaled['Close'].rolling(window=seq_len).mean()

# 2. 변동성 (Volatility) - 시장 안정성 파악
df_scaled['Volatility'] = df_scaled['Close'].rolling(window=seq_len).std()

# 3. 결측치(NaN) 제거
df_final = df_scaled.dropna()

# 데이터 확인
df_final.info()
print("="*50)
print(df_final.head())
```

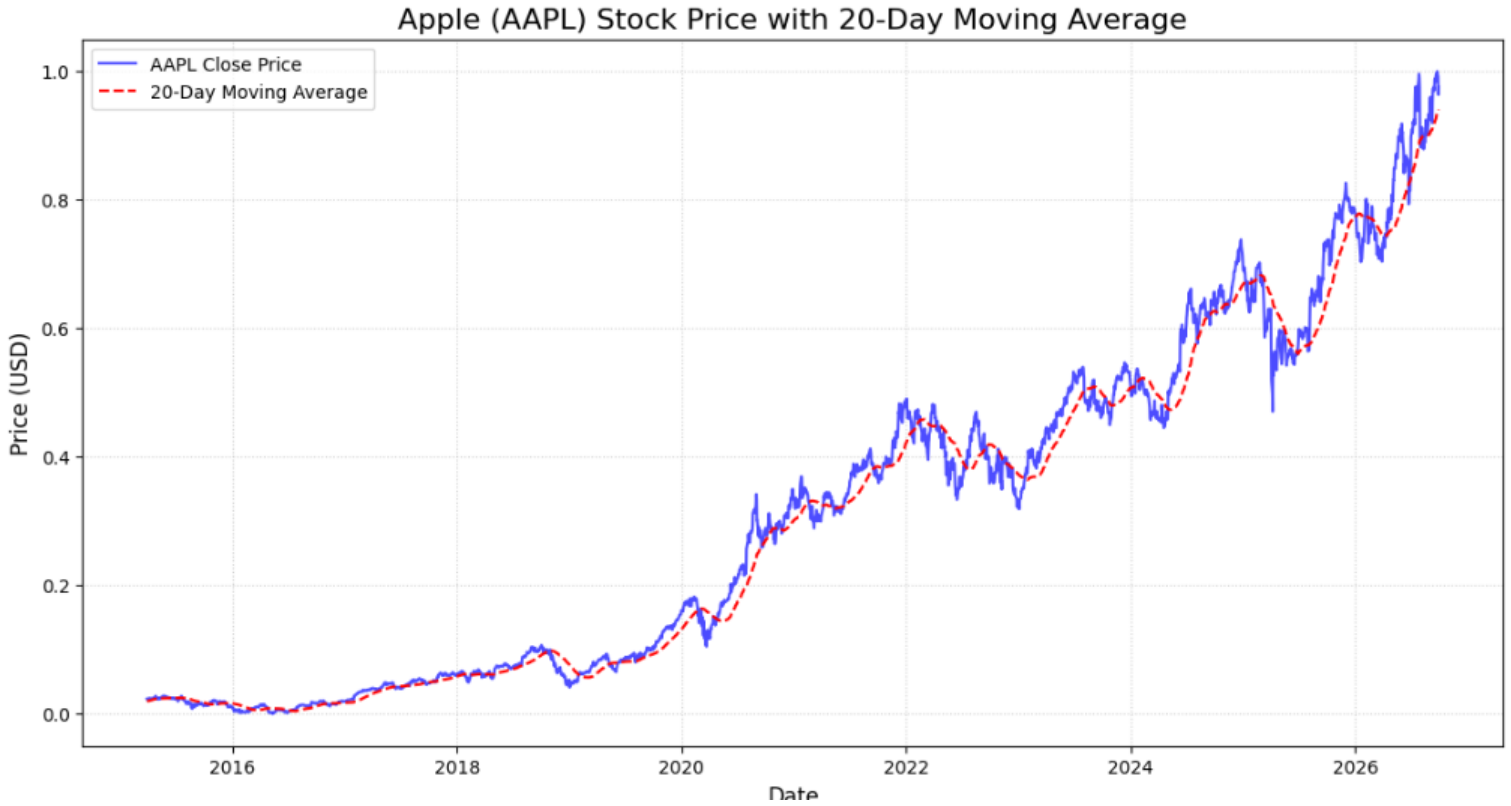
② 데이터 전처리

- 새로 생성한 특성을 그래프로 확인한다.

```
import matplotlib.pyplot as plt

# 시각화
plt.figure(figsize=(14, 7)) # 그래프 크기 설정

# 원본 종가(Close Price) 그래프
plt.plot(df_final.index,
         df_final['Close'],
         label='AAPL Close Price',
         color='blue',
         alpha=0.7)

# 20일 이동 평균선 그래프
plt.plot(df_final.index,
         df_final['MA_20'],
         label='20-Day Moving Average',
         color='red',
         linestyle='--')

# 그래프 제목 및 축 라벨 설정
plt.title('Apple (AAPL) Stock Price with 20-Day Moving Average', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price (USD)', fontsize=12)
plt.legend(fontsize=10) # 범례 표시
plt.grid(True, linestyle=':', alpha=0.6) # 그리드 라인 추가

plt.show()
```

② 데이터 전처리

- 새로 생성한 특성을 그래프로 확인한다.



③ RNN 모델 학습용 데이터셋 구축

- Sliding Window 방식으로 데이터셋 생성

```python
import numpy as np

# # 1. 시퀀스 길이(타임스텝) 설정
# 과거 60일치의 데이터를 보고 다음 날을 예측
# seq_len = 60

# 2. 빈 리스트 준비
X = [] # 학습할 문제 데이터 (입력)
y = [] # 정답 데이터 (출력)

# 3. numpy 배열로 변환하여 작업 속도 향상
data_array = df_final.to_numpy()

# 4. 정답(y)으로 사용할 'Close' 가격의 인덱스 찾기
close_idx = df_final.columns.get_loc('Close')

# 5. Sliding Window 방식으로 데이터셋 생성
for i in range(seq_len, len(data_array)):
    # i번째 시점 이전까지 seq_len일치 데이터를 X로 저장
    X.append(data_array[i-seq_len : i])

    # i번째 시점의 'Close' 가격을 y로 저장
    y.append(data_array[i, close_idx])
```

```python
# 6. 리스트를 Numpy 배열로 변환
X = np.array(X)
y = np.array(y)

# 7. 최종 데이터 형태 확인
print("--- 최종 데이터 형태 ---")
print(f"X shape: {X.shape}")
print(f"y shape: {y.shape}")
```

```
--- 최종 데이터 형태 ---
X shape: (2834, 60, 7)
y shape: (2834, 1)
```

③ RNN 모델 학습용 데이터셋 구축

- 시계열의 순서를 유지하면서 전체 데이터의 앞부분 80%는 학습용, 뒷부분 20%는 검증용으로 분할한다.

```python
# 데이터 분할

# 1. 분할 지점(split point) 계산
# 전체 데이터의 80% 지점을 계산합니다.
split_ratio = 0.8
split_index = int(len(X) * split_ratio)

# 2. 데이터 분할 (***중요: 순서를 유지하며 자릅니다***)
# 처음부터 분할 지점까지 -> 학습 데이터
X_train = X[:split_index]
y_train = y[:split_index]

# 분할 지점부터 끝까지 -> 검증 데이터
X_val = X[split_index:]
y_val = y[split_index:]

# 3. 분할된 데이터의 형태(shape) 확인
print("--- 데이터 분할 결과 ---")
print(f"학습 데이터 X shape: {X_train.shape}")
print(f"학습 데이터 y shape: {y_train.shape}")
print(f"검증 데이터 X shape: {X_val.shape}")
print(f"검증 데이터 y shape: {y_val.shape}")
```

```
--- 데이터 분할 결과 ---
학습 데이터 X shape: (2267, 60, 7)
학습 데이터 y shape: (2267, 1)
검증 데이터 X shape: (567, 60, 7)
검증 데이터 y shape: (567, 1)
```

④ SimpleRNN 모델 생성 및 학습

- SimpleRNN으로 순환신경망 모델을 생성한다.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Input

model = Sequential([
    Input(shape=(seq_len, 7)),
    SimpleRNN(units=32),
    Dense(units=1) # 다음 숫자 1개를 예측하므로 units=1
])

model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 32)	1,280
dense (Dense)	(None, 1)	33

```
Total params: 1,313 (5.13 KB)
Trainable params: 1,313 (5.13 KB)
Non-trainable params: 0 (0.00 B)
```

④ SimpleRNN 모델 생성 및 학습

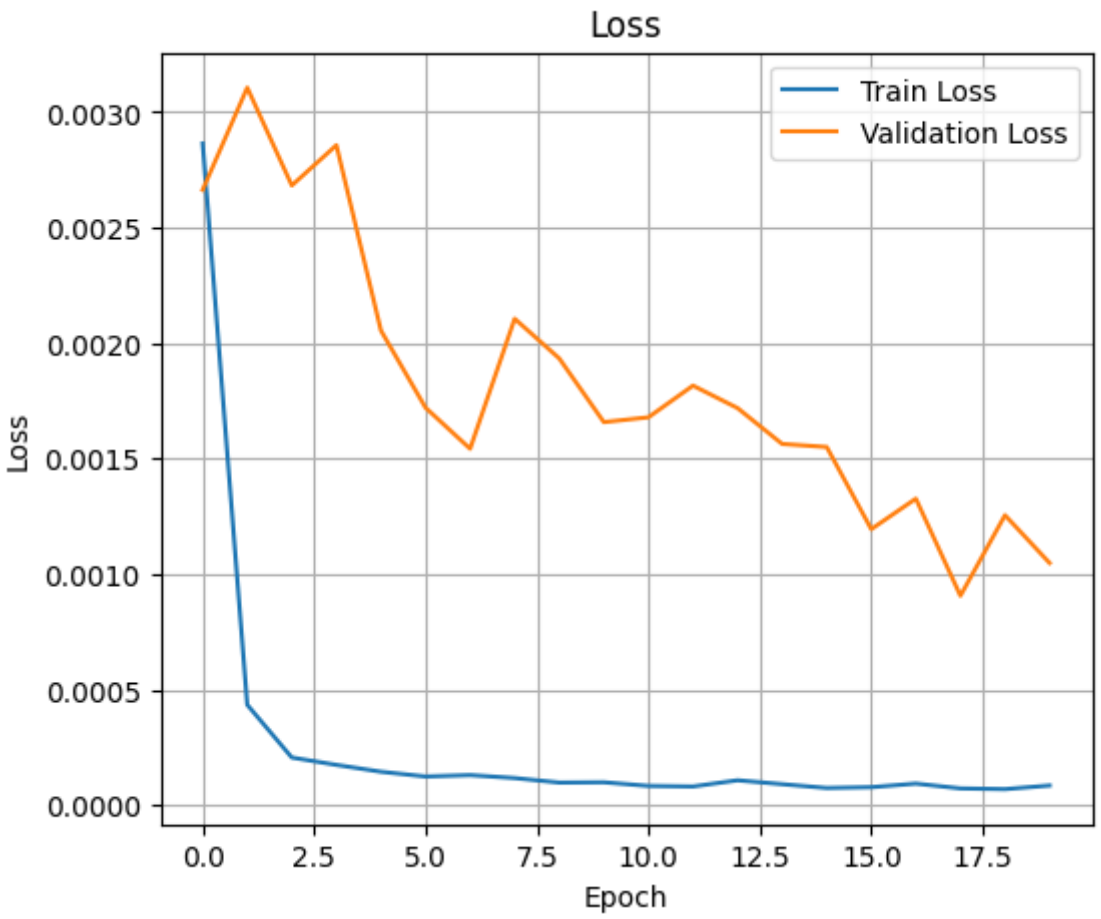
- 순환신경망 모델을 compile한 후 학습시킨다.

```
model.compile(optimizer='adam',
              loss='mean_squared_error')

history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=20)
```

⑤ 학습 결과 확인

- 손실/정확도 그래프를 그려 학습 결과를 확인한다.

```
# 학습 결과 시각화: 손실/정확도 그래프
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 5))
plt.plot(history.history['loss'],
         label='Train Loss')
plt.plot(history.history['val_loss'],
         label='Validation Loss')
plt.title('Loss')
plt.grid()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



LSTM

(Long Short-Term Memory)

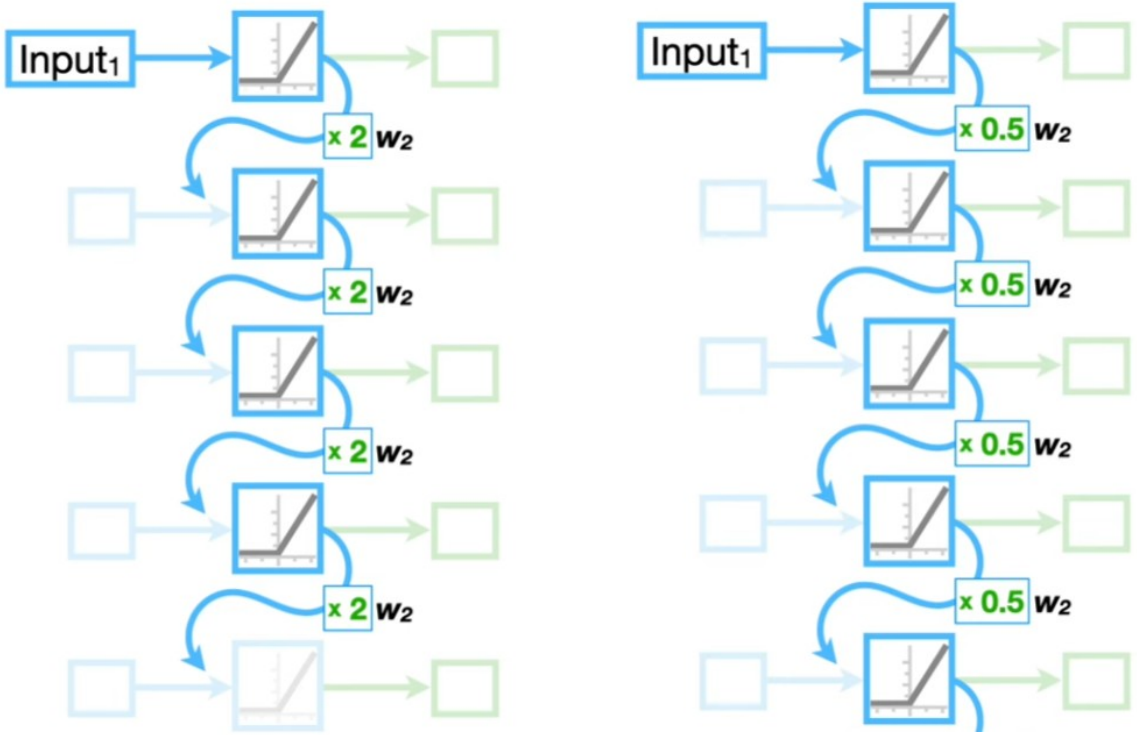
순환 신경망의 한계

- RNN은 이전 상태를 다음으로 전달하며 진행됨.

  →시퀀스가 길어질수록 **초반 정보가 희미**해짐

  →학습 과정에서 정보 전달 신호(gradient)가 점점 약해지기 때문

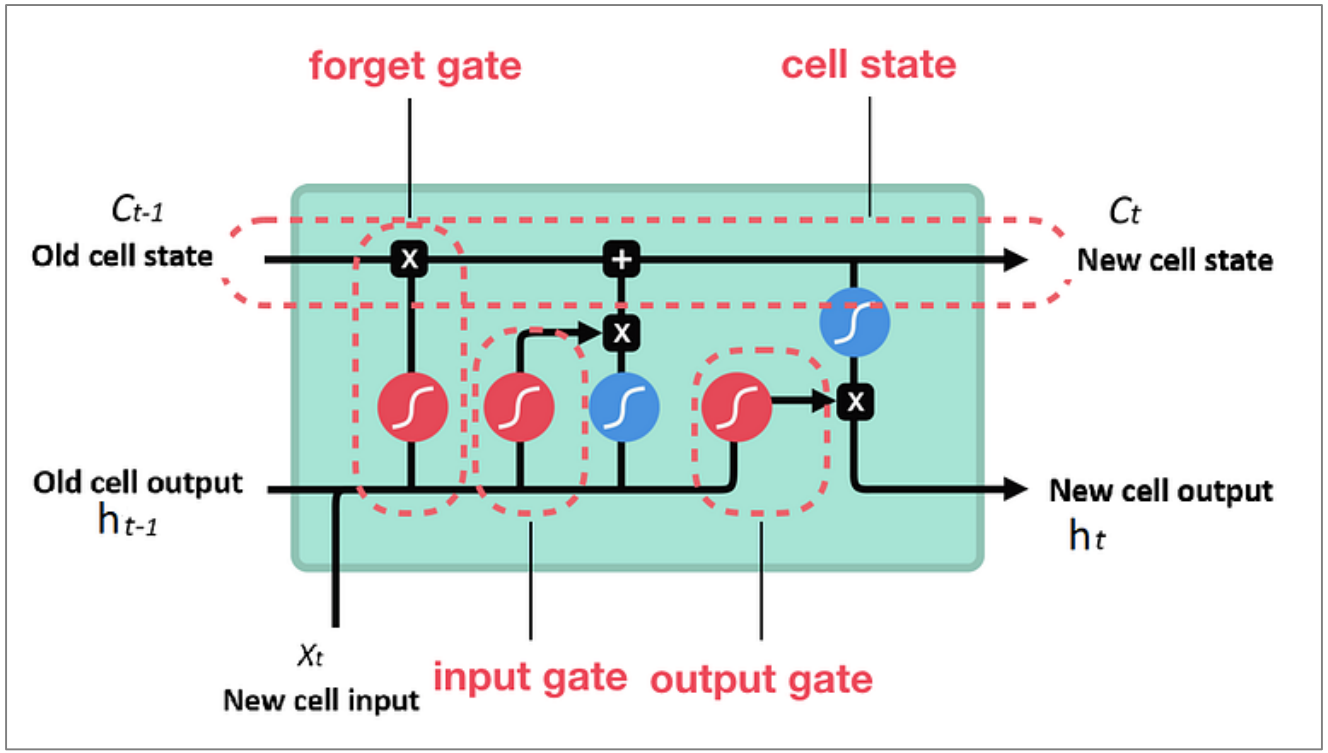
  →이를 기울기 소실 문제라고 함



https://youtu.be/AsNTP8Kwu80?si=O83AbANVkUOC4w6J

LSTM (Long Short-Term Memory)

- 순환 신경망의 한 종류로 오래 기억하는 능력을 크게 개선한 모델
- 핵심 컨셉: **모든 정보를 똑같이 기억하지 말고 중요한 것만 오래 기억하자.**



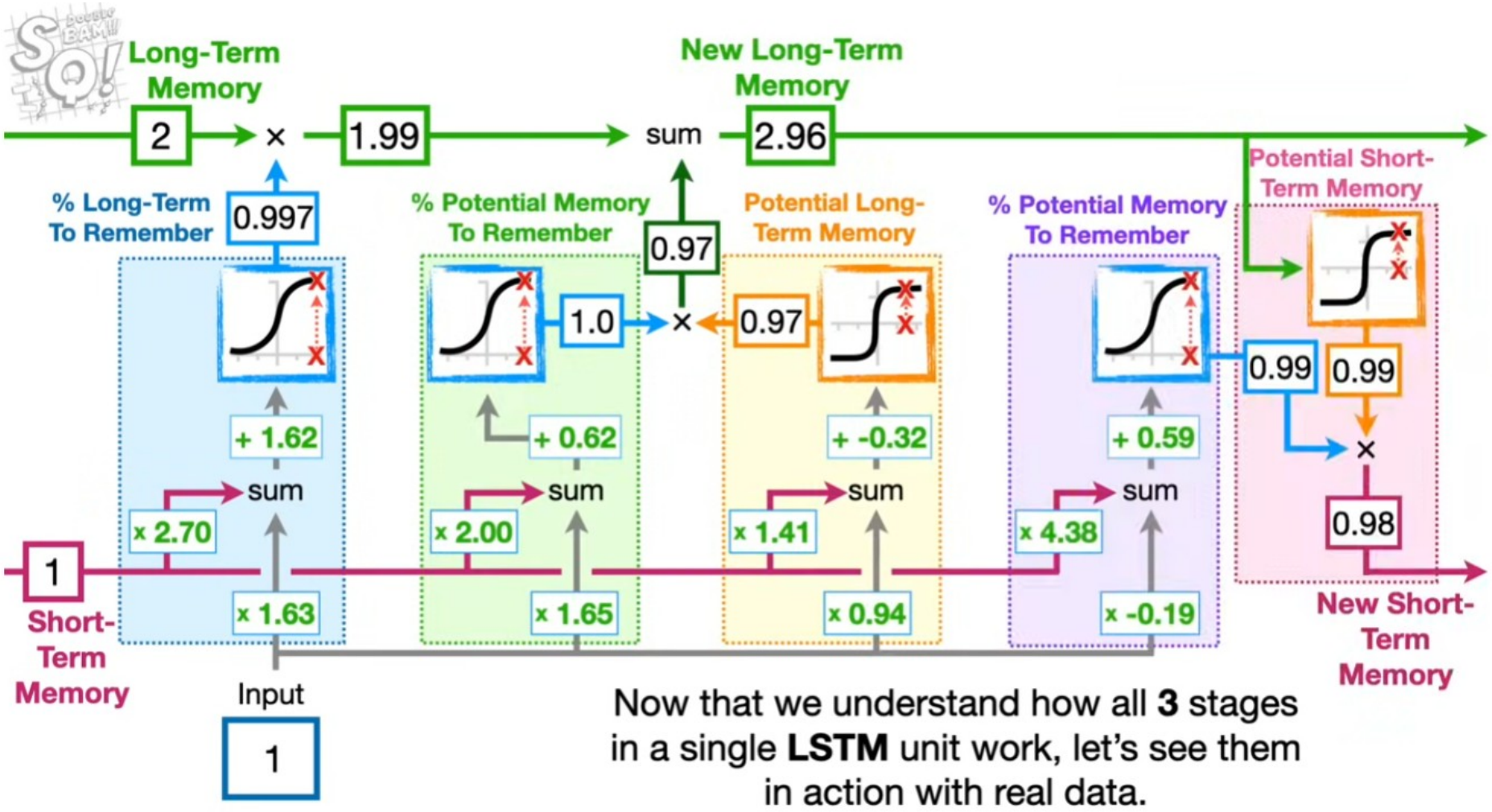
https://medium.com/%40humble_bee/rnn-recurrent-neural-networks-lstm-842ba7205bbf

LSTM의 동작 원리

- 게이트(Gate)
  - 이전 상태 정보를 얼마나 기억하고, 새로운 정보를 얼마나 받아들일지 스스로 결정

- **게이트를 통해 정보를 '선택적으로 걸러내고, 저장하고, 내보내는' 과정을 거침**
  - 망각 게이트 (Forget Gate)
    - 이전 상태의 정보 중에서 현재 시점에서 **'필요 없는 정보'를 잊어버리도록** 만듦
  - 입력 게이트 (Input Gate)
    - 현재 시점의 새로운 정보 중에서 **'기억해야 할 정보'를 선택적으로 받아들임**
  - 출력 게이트 (Output Gate)
    - 장기 기억으로 저장된 정보 중에서 '현재 시점의 결과(예측)'를 만드는 데 **필요한 정보만 내보냄**

LSTM의 동작 원리 (컴퓨터가 기억하는 방식)



https://youtu.be/YCzL96nL7j0?si=bdc05bdh1-vdPhnh

실습2-2. Apple 주가 예측하기 : LSTM

- **실습 내용:** 실습 2-1에서 처리한 Apple 주가 데이터를 LSTM 모델로 학습하기

- **실습 단계:**

  ① 데이터 준비 및 정규화

  ② 데이터 전처리

  ③ RNN 모델 학습용 데이터셋 구축

  ④ LSTM 모델 학습하기

  ⑤ 학습 결과 확인

④ LSTM 모델 생성 및 학습

- LSTM 모델 생성하기 → SimpleRNN 자리에 LSTM을 쓰면 됨

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense,Input

model = Sequential([
    Input(shape=(seq_len, 7)),
    LSTM(units=32),
    Dense(1)
])

model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 32)	5,120
dense_2 (Dense)	(None, 1)	33

Total params: 5,153 (20.13 KB)
Trainable params: 5,153 (20.13 KB)
Non-trainable params: 0 (0.00 B)

④ LSTM 모델 생성 및 학습

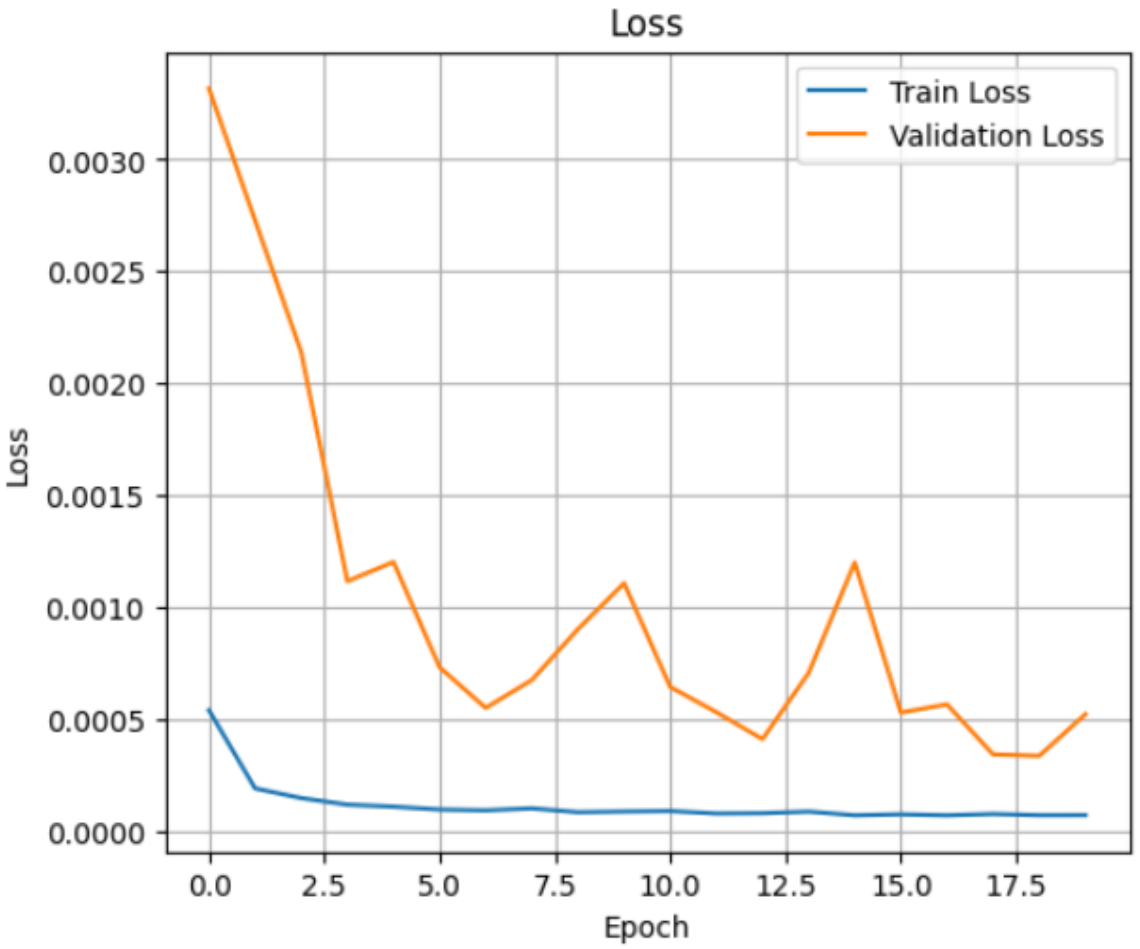
- LSTM 모델 학습하기

```
model.compile(optimizer='adam',
              loss='mean_squared_error')

history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=20)
```

⑤ 학습 결과 확인

- 결과 시각화

```
# 학습 결과 시각화: 손실/정확도 그래프
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 5))
plt.plot(history.history['loss'],
         label='Train Loss')
plt.plot(history.history['val_loss'],
         label='Validation Loss')
plt.title('Loss')
plt.grid()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



실습2-3. 하이퍼파라미터 튜닝

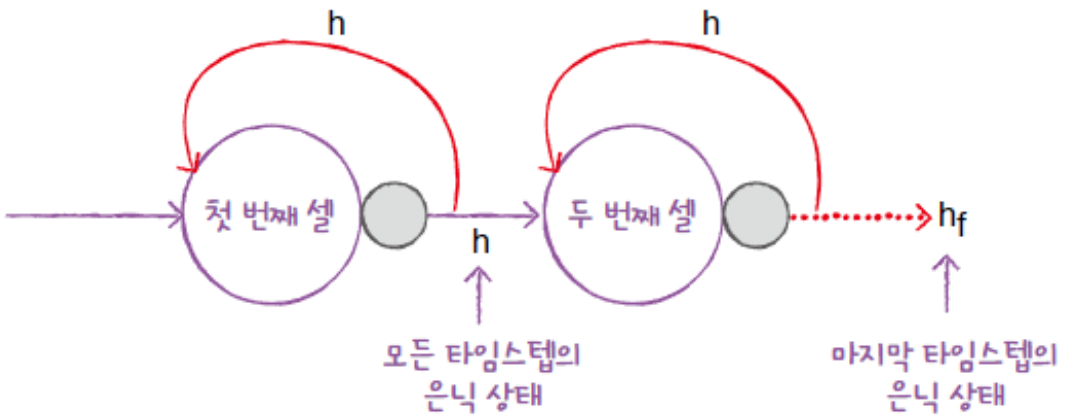
❖**실습 내용:** 애플 주식 예측 실습의 LSTM 모델에서 다음 값들을 바꿔가며 학습 결과를 관찰해보자.

- 시퀀스 크기 → 20개 / 60개
- 순환층의 개수 → 2개 / 3개
- 순환층 내의 셀 개수 → 점차 증가하게 / 동일하게 / 점차 감소하게
- Dropout의 위치와 비율 → 심층망 사이 Dropout을 0.3, 0.5 변화시켜 보기
  → 심층만 내의 Dropout을 0.3, 0.5 변화시켜 보기

2개 이상의 순환층

- RNN의 출력은,
  - 마지막 타임스텝의 은닉 상태만 출력으로 내보냄
  - 마치 입력된 시퀀스에서 읽은 모든 정보를 마지막 은닉 상태에 압축하여 전달하는 것처럼 볼 수 있음
  - 이대로 다음 순환층에 전달하면 입력의 형태가 맞지 않게 됨

- 순환층을 여러 개 쌓았을 때 셀의 출력은,
  - 첫 번째 셀은 모든 타임스텝의 은닉 상태를 출력
  - 두 번째 셀은 마지막 타임스텝의 은닉 상태만 출력



2개 이상의 순환층

- 코드에서는 첫 번째 순환층에 return_sequences=True 옵션 추가

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense, Input

model = Sequential([
    Input(shape=(seq_len, 7)),
    LSTM(units=64,
         return_sequences=True),
    LSTM(units=32),
    Dense(1)
])

model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
lstm_5 (LSTM)	(None, 60, 64)	18,432
lstm_6 (LSTM)	(None, 32)	12,416
dense_5 (Dense)	(None, 1)	33

Total params: 30,881 (120.63 KB)
Trainable params: 30,881 (120.63 KB)
Non-trainable params: 0 (0.00 B)

Dropout

- **개념: 학습할 때 일부 뉴런을 무작위로 사용하지 않는 방법**
  - 학습 과정에서 일부 뉴런의 출력을 임시로 **0으로 만들어 제외**
  - 특정 뉴런이나 특정 연결에 지나치게 의존하는 것을 방지
  - 여러 뉴런이 고르게 학습되도록 도움

- **과적합(Overfitting)을 줄이기 위해 사용**
  - 학습 데이터에만 지나치게 맞춰지는 현상 완화
  - 새로운 데이터에 대한 **일반화 성능 향상**
  - 모델이 특정 특징에만 의존하지 않도록 함

※ Dropout은 **학습할 때만 적용**되고, 예측할 때는 모든 뉴런을 사용

Dropout

- 코드에서 사용하기

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense, Input

model = Sequential([
    Input(shape=(seq_len, 7)),
    LSTM(units=64,
         return_sequences=True),
    Dropout(0.2),    # 20%의 뉴런을 랜덤하게 비활성화하여 과적합 방지
    LSTM(units=32),
    Dense(1)
])
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
lstm_7 (LSTM)	(None, 60, 64)	18,432
dropout_1 (Dropout)	(None, 60, 64)	0
lstm_8 (LSTM)	(None, 32)	12,416
dense_6 (Dense)	(None, 1)	33

Total params: 30,881 (120.63 KB)
Trainable params: 30,881 (120.63 KB)
Non-trainable params: 0 (0.00 B)

Dropout

- dropout을 어디에 어떻게 설정하느냐에 따라 적용 대상에 차이가 있음

구분	Dropout 층	순환 층의 dropout	순환 층의 recurrent_dropout
적용 대상	이전 층의 **출력**	순환 층의 **입력**	순환 층의 **순환 연결**
사용 방식	별도의 층으로 추가	층 내부의 파라미터로 설정	층 내부의 파라미터로 설정
주요 역할	층 간의 과적합 방지	순차 데이터의 입력 노이즈 방지	순환 학습 과정의 과적합 방지

- 각 위치에서의 사용 예시

```
model = Sequential([
        Input(shape=(seq_len, 7)),
        LSTM(units=64,
            return_sequences=True,
            dropout=0.3,
            recurrent_dropout=0.2),
        Dropout(0.2),
        LSTM(units=32),
        Dense(1)
])
```

```
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
lstm_11 (LSTM)	(None, 60, 64)	18,432
dropout_2 (Dropout)	(None, 60, 64)	0
lstm_12 (LSTM)	(None, 32)	12,416
dense_8 (Dense)	(None, 1)	33

```
Total params: 30,881 (120.63 KB)
Trainable params: 30,881 (120.63 KB)
Non-trainable params: 0 (0.00 B)
```

모델 저장을 위한
콜백과 조기 종료

모델 저장과 복원

- save_weights() 함수
  : 훈련된 모델의 파라미터를 저장하는 함수

- save() 함수
  : 모델의 구조와 파라미터를 함께 저장하는 함수

```
# 모델 파라미터(가중치)만 저장
model_saving.save_weights('model.weights.h5')

# 모델 전체 구조 및 파라미터 저장
model_saving.save('my_model.keras')
```

모델 저장과 복원

- load_weights() 함수
  : 저장된 모델의 파라미터를 불러오는 함수

- load_model() 함수
  : 저장된 모델의 구조와 파라미터를 불러오는 함수

```
# 저장되어있는 모델 파라미터를 가져온 후, 가져온 파라미터 값 확인
model_loading.load_weights('model.weights.h5')
model_loading.get_weights()
```

```
# 저장되어있는 모델을 통채로 로딩하고, 구조를 확인
model_load_all = keras.models.load_model('my_model.keras')
model_load_all.summary()
```

콜백과 조기 종료

- 콜백(Callback)
  - 특정 이벤트가 발생했을 때 호출되도록 미리 정의된 함수
  - 개발자가 함수를 호출하는 것이 아니라, 시스템(프레임워크)이 정해진 시점에 대신 호출해주는 함수

- 딥러닝에서의 콜백
  - 모델이 학습되는 동안 발생하는 다양한 이벤트를 모니터링
  - 특정 조건에 따라 자동으로 작업을 수행

콜백과 조기 종료

- 콜백 사용하기
  - keras.callbacks 모듈에 다양한 콜백 함수가 내장되어 있음
  - 콜백은 model.fit() 함수의 callbacks 매개변수에 리스트 형태로 전달하여 사용

- 꼭 알아둘 콜백 함수: ModelCheckpoint 와 EarlyStopping
  - EarlyStopping : 학습을 조기에 멈춤
  - ModelCheckpoint : 학습이 멈춘 시점과는 별개로, 가장 좋았던 성능을 기록한 시점의 모델을 저장

콜백 함수 : ModelCheckpoint

- ModelCheckpoint
  - 모델의 학습 과정에서 가장 좋은 성능을 보인 시점의 가중치(weights) 또는 모델 전체를 파일로 자동 저장하는 콜백

  - 주요 매개변수:
    - filepath: 저장될 파일의 경로와 이름을 지정
      - 예: 'best_model.h5'
    - monitor: 모니터링할 지표를 지정
      - 예: 'val_loss', 'val_accuracy'
    - save_best_only: True로 설정하면 모니터링 지표가 최고 기록을 경신했을 때만 모델을 저장
    - mode: 'min'(최소값), 'max'(최대값) 중 하나를 설정하여 지표의 목표 방향을 정함
      - 예: val_loss는 min, val_accuracy는 max

콜백 함수 : ModelCheckpoint

- ModelChechpoint 콜백 사용 예시

```
# ModelCheckpoint 콜백 생성.
# save_best_only=True :최고의 성능을 낸 에포크의 모델 파라미터를 저장함
checkpoint_cb = keras.callbacks.ModelCheckpoint('best-model1.keras', save_best_only=True)

# 에포크 20번을 다 함
history_cbmodel1 = model_callback1.fit(train_scaled, train_target, epochs=20,
                    validation_data=(val_scaled, val_target),
                    callbacks=[checkpoint_cb]) # 콜백 등록
```

콜백 함수 : EarlyStopping

- EarlyStopping
  - 모델의 성능 개선이 일정 기간 동안 없을 때 학습을 자동으로 중단하는 콜백
  - 주요 매개변수:
    - monitor: 모니터링할 지표를 지정
      - 예: 'val_loss'
    - patience: 성능 개선이 없는 에포크를 몇 번까지 기다릴지 지정
      - 예: patience=5는 5 에포크 동안 val_loss가 감소하지 않으면 학습을 중단
    - restore_best_weights: True로 설정하면 학습이 중단될 때, 가장 성능이 좋았던 시점의 가중치로 모델을 복원해줌

콜백 함수 : EarlyStopping

- ModelChechpoint + EarlyStopping 콜백 사용 예시

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

model = Sequential([
    LSTM(units=64, return_sequences=True, input_shape=(SEQUENCE_LENGTH, 7)),
    LSTM(units=32),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

checkpoint_cb = ModelCheckpoint('best_model.keras', save_best_only = True)
early_stopping_cb = EarlyStopping(patience=3, restore_best_weights = True)

history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
                    epochs=100, verbose=0,
                    callbacks = [checkpoint_cb, early_stopping_cb])
```

시계열 데이터의 전처리

결측치 처리와 데이터 평활화

결측치 처리

- 시계열 데이터의 결측치 처리 방법

Forward Fill (ffill)	- 이전 시점의 값으로 채움 - 데이터 변화가 크지 않을 때 유용
Backward Fill (bfill)	- 다음 시점의 값으로 채움
선형 보간 (Linear Interpolation)	- 누락된 지점의 양옆 값을 이용해 직선을 그려 중간값을 추정 - 부드러운 변화를 가정할 때 효과적

결측치 처리

- Forward Fill (ffill)

```
df['price'].fillna(method = 'ffill', inplace = True)
```

	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	NaN
2	2021-01-03	apple	NaN
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	NaN
7	2021-01-04	mango	2.8

fill

fill

fill

	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	0.8
2	2021-01-03	apple	0.8
3	2021-01-04	apple	1.2
4	2021-01-01	mango	1.2
5	2021-01-02	mango	3.1
6	2021-01-03	mango	3.1
7	2021-01-04	mango	2.8

→ 최근에는 아래 방식이 권장됨

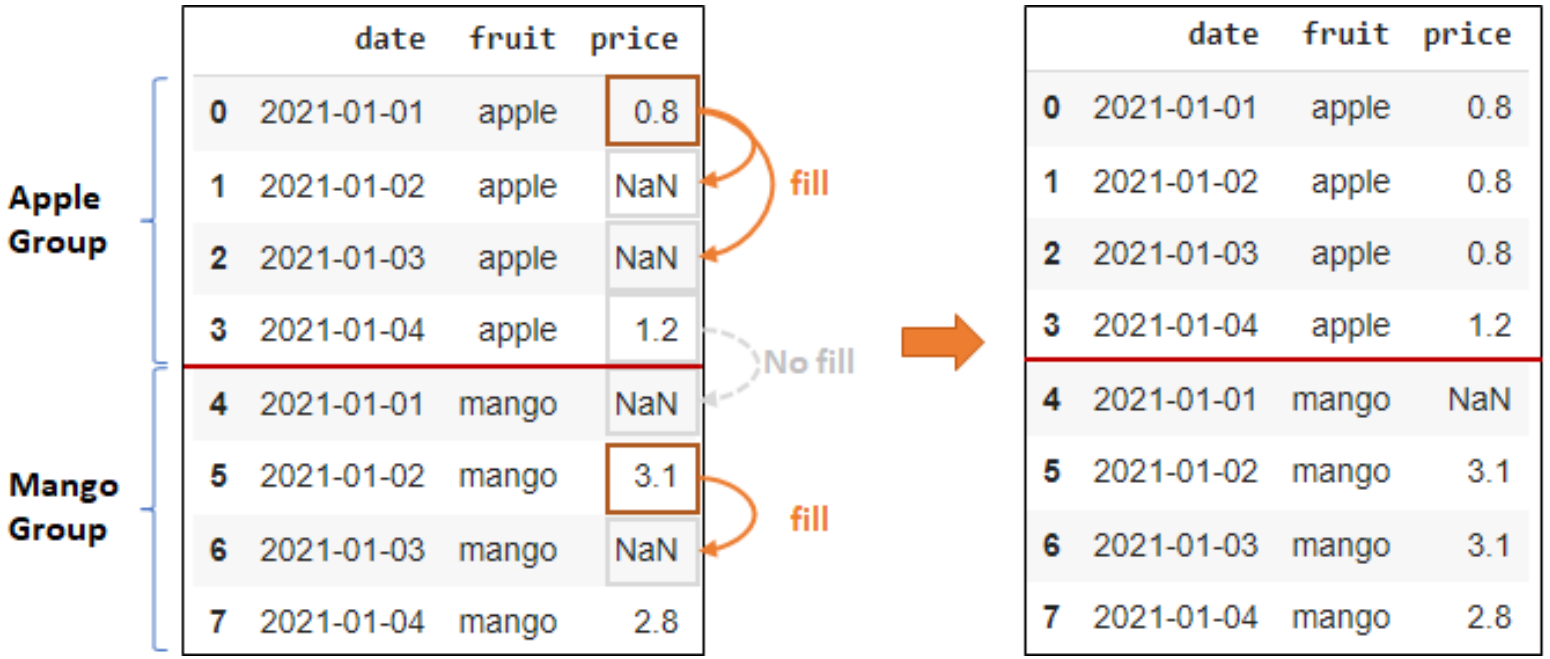
```
df['price'] = df['price'].ffill()
```

https://towardsdatascience.com/how-to-fill-missing-data-with-pandas-8cb875362a0d/

결측치 처리

- **그룹별로** Forward Fill (ffill) 적용하기

```
df['price'] = df.groupby('fruit')['price'].ffill()
```



	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	NaN
2	2021-01-03	apple	NaN
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	NaN
7	2021-01-04	mango	2.8

	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	0.8
2	2021-01-03	apple	0.8
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	3.1
7	2021-01-04	mango	2.8

결측치 처리

- Backward Fill (bfill)

```
df['price'].fillna(method = 'bfill', inplace = True)
```

	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	NaN
2	2021-01-03	apple	NaN
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	NaN
7	2021-01-04	mango	2.8

fill

fill

fill

	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	1.2
2	2021-01-03	apple	1.2
3	2021-01-04	apple	1.2
4	2021-01-01	mango	3.1
5	2021-01-02	mango	3.1
6	2021-01-03	mango	2.8
7	2021-01-04	mango	2.8

→ 최근에는 아래 방식이 권장됨

```
df['price'] = df['price'].bfill()
```

https://towardsdatascience.com/how-to-fill-missing-data-with-pandas-8cb875362a0d/

결측치 처리

- 선형 보간 (Linear Interpolation)

```
df['price'].interpolate(method = 'linear', inplace = True)
```

	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	NaN
2	2021-01-03	apple	NaN
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	NaN
7	2021-01-04	mango	2.8

interpolate

$$\frac{1.2 - 0.8}{3} = 0.133$$

interpolate

$$\frac{3.1 - 1.2}{2} = 0.95$$

interpolate

$$\frac{2.8 - 3.1}{2} = -0.15$$

	date	fruit	price
0	2021-01-01	apple	0.800
1	2021-01-02	apple	0.933
2	2021-01-03	apple	1.067
3	2021-01-04	apple	1.200
4	2021-01-01	mango	2.150
5	2021-01-02	mango	3.100
6	2021-01-03	mango	2.950
7	2021-01-04	mango	2.800

+0.133

+0.133

+0.95

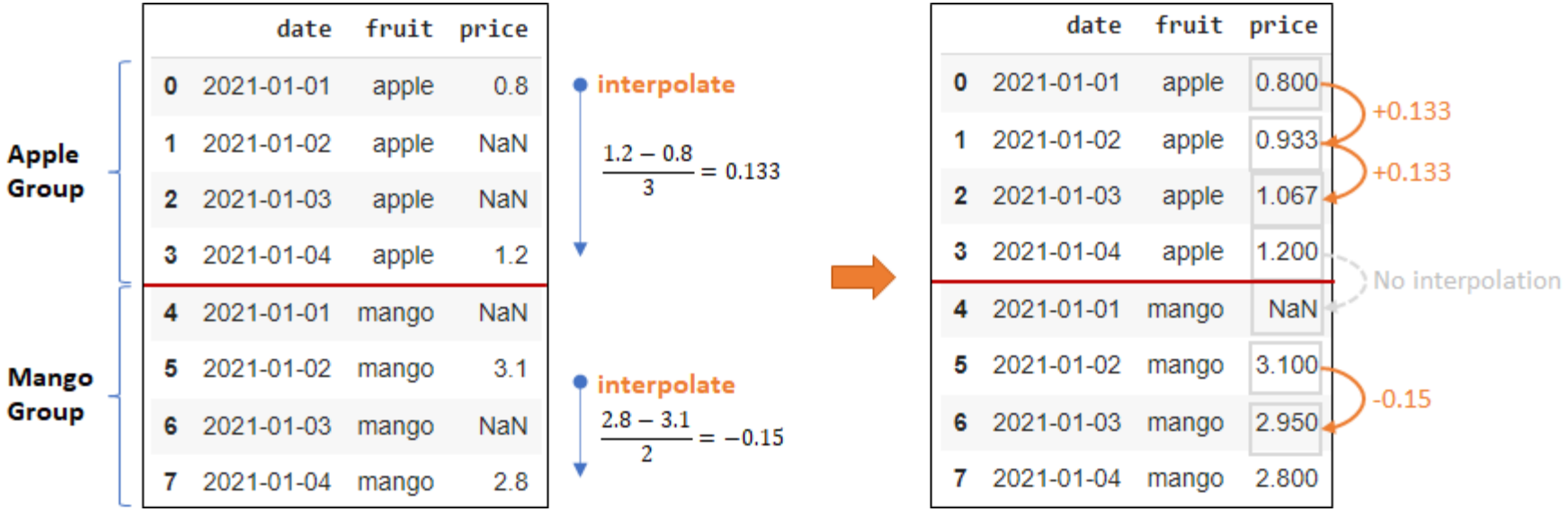
-0.15

https://towardsdatascience.com/how-to-fill-missing-data-with-pandas-8cb875362a0d/

결측치 처리

- **그룹별로** 선형 보간 (Linear Interpolation)

```
df.groupby('fruit')['price'].apply(lambda x: x.interpolate(method='linear'))
```



	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	NaN
2	2021-01-03	apple	NaN
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	NaN
7	2021-01-04	mango	2.8

	date	fruit	price
0	2021-01-01	apple	0.800
1	2021-01-02	apple	0.933
2	2021-01-03	apple	1.067
3	2021-01-04	apple	1.200
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.100
6	2021-01-03	mango	2.950
7	2021-01-04	mango	2.800

https://towardsdatascience.com/how-to-fill-missing-data-with-pandas-8cb875362a0d/

실습 데이터: NASA 터보엔진 데이터셋

- 데이터셋 다운로드
  - https://www.Kaggle.com/datasets/behrad3d/nasa-cmaps

- 데이터셋 구성
  - 4개의 서브셋(FD001, FD002, FD003, FD004)으로 구성
    - train_FD001.txt: 훈련 데이터 (100개의 엔진)
    - test_FD001.txt: 테스트 데이터 (100개의 엔진)
    - RUL_FD001.txt: 테스트 데이터의 실제 RUL 값

- 데이터 구성
  - **열 1**: 엔진 ID
  - **열 2**: 사이클(시간)
  - **열 3 ~ 5**: 운영 설정 1~3
  - **열 6 ~ 26**: 21개의 센서 데이터

결측치 처리 실습

- 데이터 로딩

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

# 1. 데이터 불러오기
columns_name = ['unit_number', 'cycles', 'op_setting_1', 'op_setting_2', 'op_setting_3'] + \
                [ f'sensor_{i}' for i in range(1, 22) ]

df = pd.read_csv('/content/drive/MyDrive/_PdM/NASA/train_FD001.txt', sep=r'\s+', header=None, names=columns_name)
```

```
# 2. 1번 엔진의 'sensor_2' 데이터만 따로 복사하여 실습 진행
engine_1_sensor_2 = df[df['unit_number'] == 1][['cycles', 'sensor_2']].copy()
```

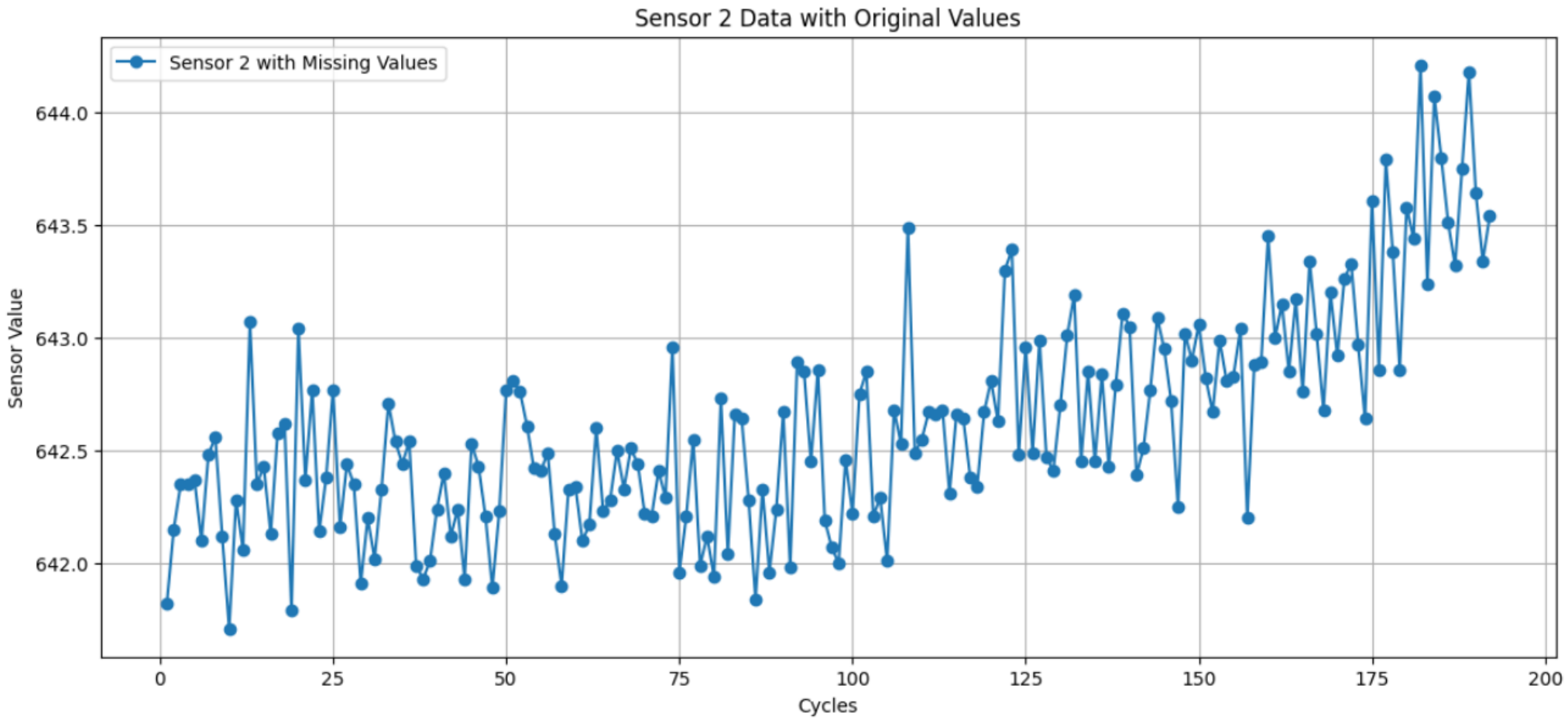
결측치 처리 실습

- 원본 데이터 확인

```
# 원본 데이터 확인 (시각화)
plt.figure(figsize=(14, 6))
plt.plot(engine_1_sensor_2['cycles'], engine_1_sensor_2['sensor_2'], 'o-', label='Sensor 2 with Missing Values')
plt.title('Sensor 2 Data with Original Values')
plt.xlabel('Cycles')
plt.ylabel('Sensor Value')
plt.legend()
plt.grid(True)
plt.show()
```

결측치 처리 실습

- 원본 데이터 확인



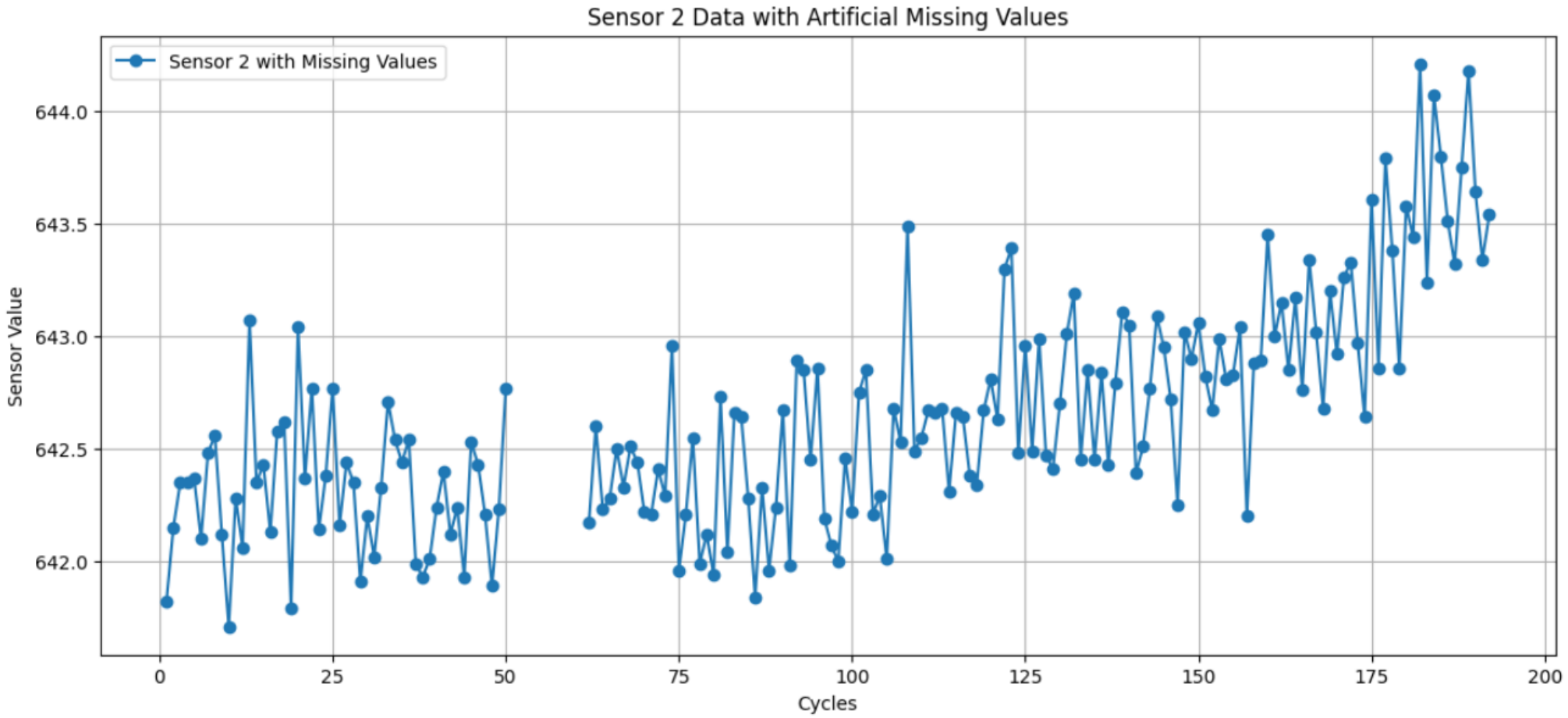
결측치 처리 실습

- 인위적으로 결측치 만들기

```
# 3. 인위적으로 결측치 만들기 (예: 50~60 사이클 구간의 데이터를 삭제)
engine_1_sensor_2.loc[50:60, 'sensor_2'] = np.nan

# 4. 결측치 확인 (시각화)
plt.figure(figsize=(14, 6))
plt.plot(engine_1_sensor_2['cycles'], engine_1_sensor_2['sensor_2'], 'o-', label='Sensor 2 with Missing Values')
plt.title('Sensor 2 Data with Artificial Missing Values')
plt.xlabel('Cycles')
plt.ylabel('Sensor Value')
plt.legend()
plt.grid(True)
plt.show()
```

결측치 처리 실습

- 인위적으로 결측치 만든 결측치 확인



결측치 처리 실습

- 결측치 처리
  - Forward Fill (ffill) : 이전 값으로 채우기
  - 선형 보간 (linear interpolate) : 누락된 지점의 양옆 값을 이용해 직선을 그려 중간값을 추정

```
# --- 이제 결측치를 채워봅니다 ---

# 5-1. Forward Fill 적용
engine_1_sensor_2['sensor_2_ffill'] = engine_1_sensor_2['sensor_2'].ffill()

# 5-2. 선형 보간 적용
engine_1_sensor_2['sensor_2_interpolated'] = engine_1_sensor_2['sensor_2'].interpolate(method='linear')
```

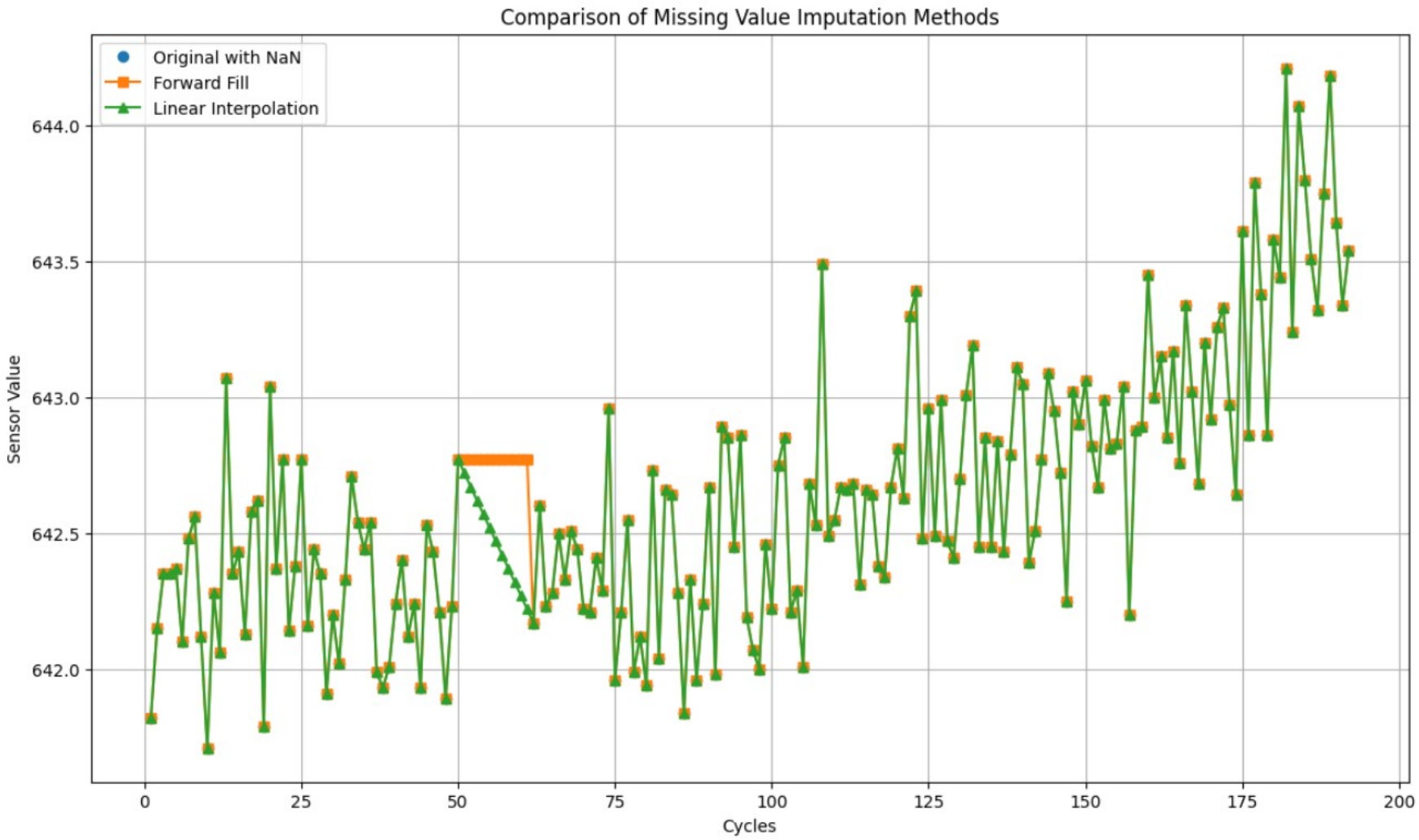
결측치 처리 실습

- 결측치 처리 결과 확인

```
# 6. 결과 비교 시각화
plt.figure(figsize=(14, 8))
# 원본 + 결측치
plt.plot(engine_1_sensor_2['cycles'], engine_1_sensor_2['sensor_2'], 'o', label='Original with NaN')
# ffill 결과
plt.plot(engine_1_sensor_2['cycles'], engine_1_sensor_2['sensor_2_ffill'], 's-', label='Forward Fill')
# 선형 보간 결과
plt.plot(engine_1_sensor_2['cycles'], engine_1_sensor_2['sensor_2_interpolated'], '^-', label='Linear Interpolation')

plt.title('Comparison of Missing Value Imputation Methods')
plt.xlabel('Cycles')
plt.ylabel('Sensor Value')
plt.legend()
plt.grid(True)
plt.show()
```

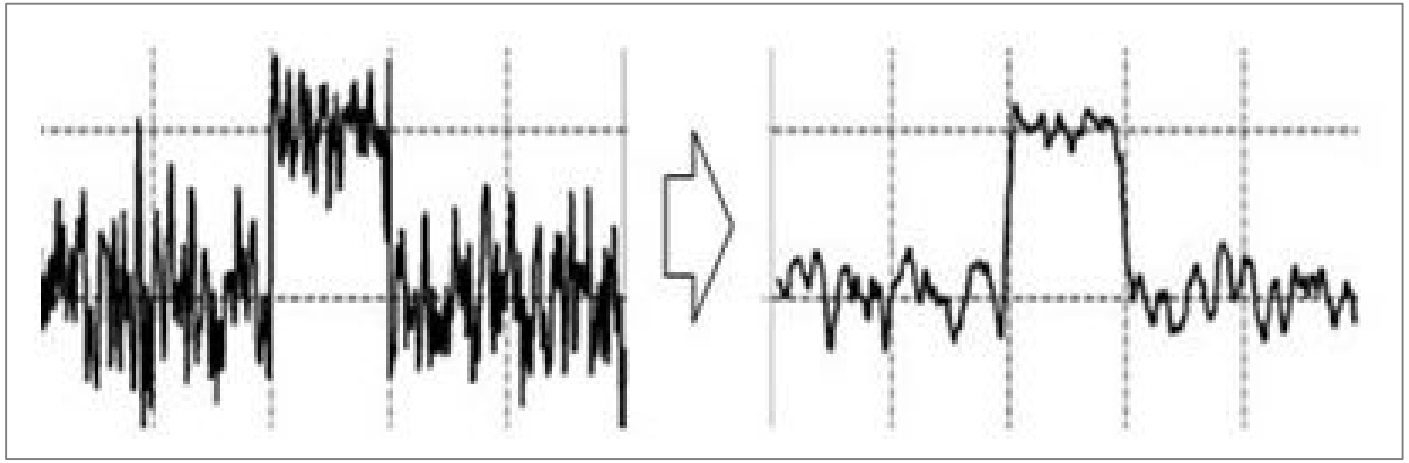
결측치 처리 실습

- 결측치 처리 결과 확인



노이즈 제거

- 필요성
  - 센서 데이터에는 장비의 실제 상태 변화와 상관없는 미세한 노이즈(Noise)가 섞여 있는 경우가 많음
  - 이러한 노이즈는 모델의 학습을 방해할 수 있음
- 이동 평균법(Moving Average)
  - 단기적인 변동(노이즈)을 완화시켜 데이터의 장기적인 추세를 파악하는 전처리 기법

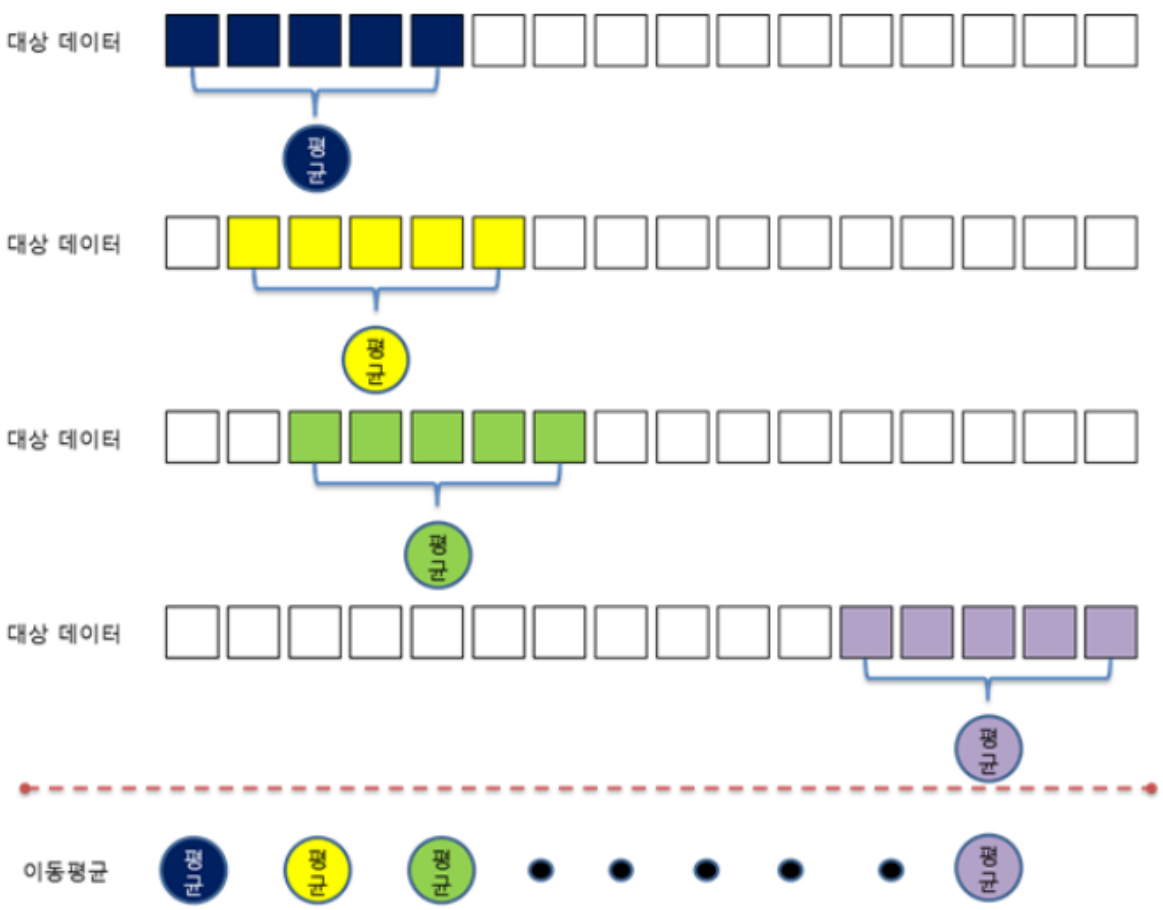


https://namu.wiki/w/%EC%9D%B4%EB%8F%99%ED%8F%89%EA%B7%A0%EC%84%A0 http://www.ktword.co.kr/test/view/view.php?no=3665

노이즈 제거

- 이동 평균법의 원리
  - "움직이는 창문(Sliding Window)"을 데이터 위로 이동시킴
  - 창문 안에 있는 데이터들의 평균을 계산하여 현재 값을 대체

- 중요한 것은 Window Size!



https://m.blog.naver.com/ntkor/199891958

노이즈 제거 실습 : rolling 함수 활용

- Window 크기에 따라 결과 그래프가 어떻게 변하는지 확인해보자

```
# 1. 1번 엔진의 'sensor_7' 데이터 사용 (변동성이 비교적 큰 센서)
engine_1_sensor_7 = df[df['unit_number'] == 1][['cycles', 'sensor_7']].copy()

# 2. 다양한 윈도우 크기로 이동 평균 계산
window_size_small = 5
window_size_large = 20

engine_1_sensor_7['rolling_mean_small'] = engine_1_sensor_7['sensor_7'].rolling(window=window_size_small).mean()
engine_1_sensor_7['rolling_mean_large'] = engine_1_sensor_7['sensor_7'].rolling(window=window_size_large).mean()
```

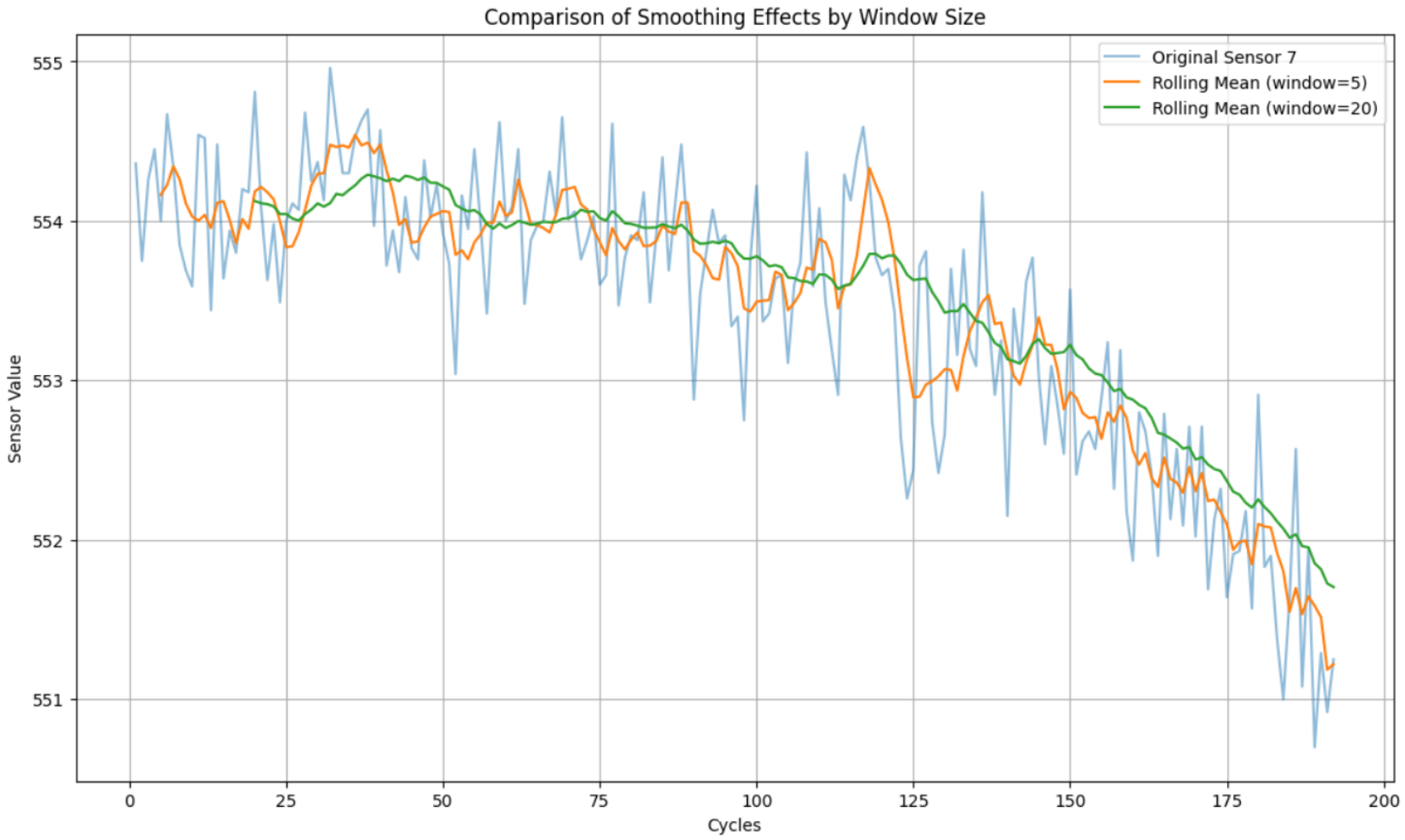
노이즈 제거 실습 : rolling 함수 활용

- Window 크기에 따라 결과 그래프가 어떻게 변하는지 확인해보자

```
# 3. 결과 비교 시각화
plt.figure(figsize=(14, 8))
# 원본 데이터
plt.plot(engine_1_sensor_7['cycles'], engine_1_sensor_7['sensor_7'], label='Original Sensor 7', alpha=0.5)
# 작은 윈도우 이동 평균
plt.plot(engine_1_sensor_7['cycles'], engine_1_sensor_7['rolling_mean_small'], label=f'Rolling Mean (window={window_size_small})')
# 큰 윈도우 이동 평균
plt.plot(engine_1_sensor_7['cycles'], engine_1_sensor_7['rolling_mean_large'], label=f'Rolling Mean (window={window_size_large})')

plt.title('Comparison of Smoothing Effects by Window Size')
plt.xlabel('Cycles')
plt.ylabel('Sensor Value')
plt.legend()
plt.grid(True)
plt.show()
```

노이즈 제거 실습 : rolling 함수 활용

- Window 크기에 따라 결과 그래프가 어떻게 변하는지 확인해보자



노이즈 제거 실습 고찰

- 어떤 Window 크기가 최적인가?
  - 정답은 없으며, 분석의 목적과 데이터의 특성에 따라 달라짐
  - **Window가 작으면**
    - 노이즈 제거 효과가 적지만, 데이터의 급격한 변화를 잘 포착
  - **Window가 크면**
    - 노이즈 제거 효과가 크고(매우 부드러운 선), 데이터의 세부적인 변화는 놓칠 수 있음
  - **적용 포인트:** 미세한 이상 징후를 잡으려면 작은 윈도우, 전반적인 트렌드를 보려면 큰 윈도우

시계열 데이터의 특성공학

롤링 통계와 지연 특징

롤링 통계 : 데이터의 흐름에서 패턴 찾기

- 개념
  - 시간에 따라 변화하는 데이터의 노이즈를 줄이고, 숨겨진 추세를 파악하기 위한 기법
  - 시간의 흐름에 따른 데이터의 국소적인(Local) 특성 변화를 파악할 수 있음

- 원리
  ① 움직이는 창문(Sliding Window)을 설정
  ② 창문(Window) 안에 들어온 데이터들의 통계치를 계산
  ③ 새로운 특성으로 만듦

- 예지보전에서의 의미
  - 반도체 장비의 마모 과정은 시간에 따른 점진적인 변화로 나타남
  - 이 변화의 '흐름'을 포착하는 것이 예지 보전의 핵심임

롤링 통계 : 데이터의 흐름에서 패턴 찾기

- 주요 통계량

이동 평균 (Rolling Mean)	- 윈도우 내 데이터의 평균 - 단기적인 변동을 제거하고 데이터의 장기적인 추세를 부드럽게 보여줌 - "센서 값이 전반적으로 상승하고 있는가?"
이동 표준편차 (Rolling Standard Deviation)	- 윈도우 내 데이터의 변동성 - 데이터의 변동성 또는 안정성을 보여줌 - 이 값이 갑자기 커진다면 장비가 불안정한 상태임을 의미할 수 있음 - "장비가 고장 나기 직전에 센서 값이 불안정하게 흔들리지 않는가?"
이동 최댓값/최솟값 (Rolling Max/Min)	- 윈도우 내 극단값을 포착함

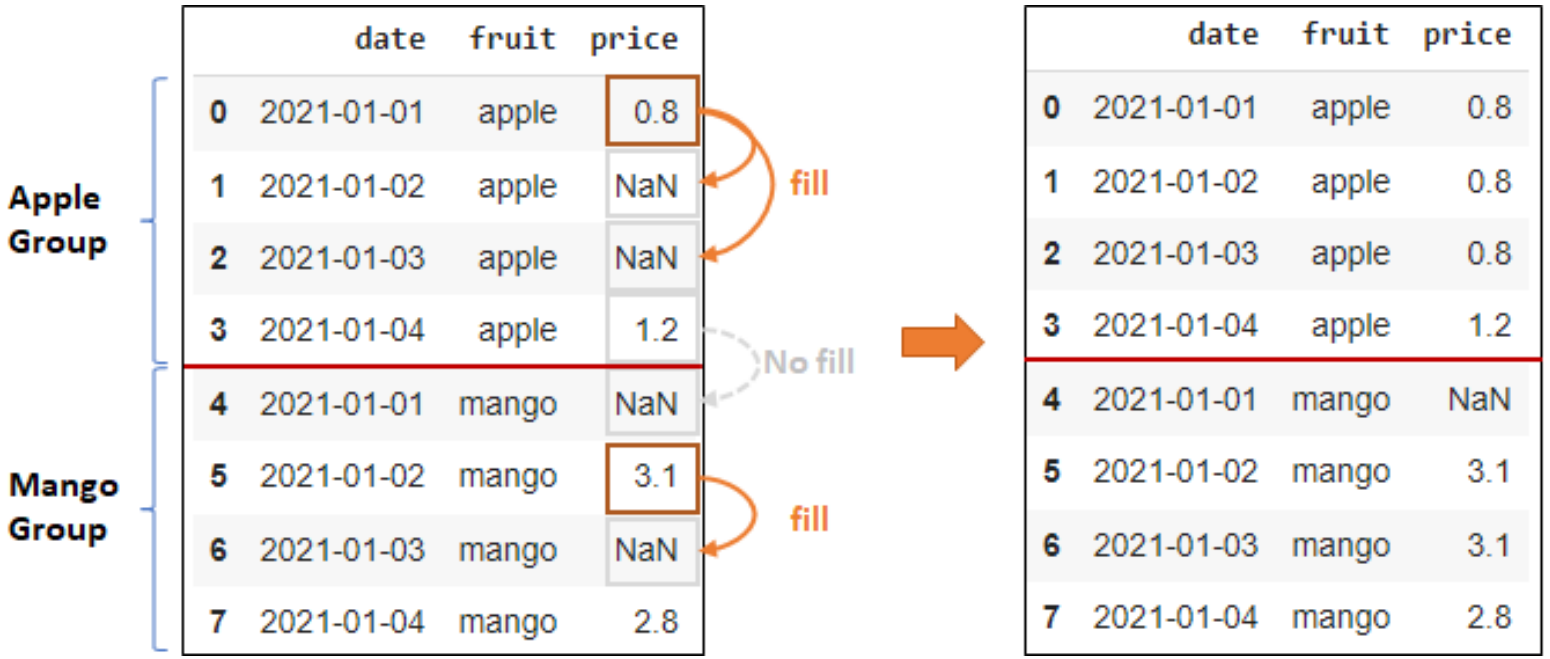
목적에 따라 달리 사용되는 이동평균 이해하기

- 데이터를 깨끗하게 만들 것인가? → **전처리**
- 모델에게 더 많은 정보를 줄 것인가? → **특성 공학**

구분	전처리 (Preprocessing)	특성 공학 (Feature Engineering)
목표	데이터 **정제** 및 노이즈 제거	모델을 위한 **정보(힌트) 추가**
결과물	부드러워진 **단일 시계열**	원본 + **새로운 특성(열)**
데이터 처리	원본 데이터를 **대체**	원본 데이터 옆에 **추가**
핵심 질문	"데이터의 진짜 추세는 무엇인가?"	"이 데이터의 단기적 추세는 예측에 도움이 되는가?"

롤링 통계 : 그룹별로 rolling() 함수 적용하기

```
df['price'] = df.groupby('fruit')['price'].ffill()
```



	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	NaN
2	2021-01-03	apple	NaN
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	NaN
7	2021-01-04	mango	2.8

	date	fruit	price
0	2021-01-01	apple	0.8
1	2021-01-02	apple	0.8
2	2021-01-03	apple	0.8
3	2021-01-04	apple	1.2
4	2021-01-01	mango	NaN
5	2021-01-02	mango	3.1
6	2021-01-03	mango	3.1
7	2021-01-04	mango	2.8

롤링 통계 실습: Pandas rolling() 함수 활용

- 데이터 로딩

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

# 1. 데이터 불러오기
columns_name = ['unit_number', 'cycles', 'op_setting_1', 'op_setting_2', 'op_setting_3'] + \
                [ f'sensor_{i}' for i in range(1, 22) ]
df = pd.read_csv('/content/drive/MyDrive/_PdM/NASA/train_FD001.txt', sep=r'\s+', header=None, names=columns_name)
```

롤링 통계 실습: Pandas rolling() 함수 활용

- 롤링 통계 계산

```
# 2. 롤링 통계를 계산할 윈도우 크기 설정
window_size = 10

# 3. 이동 평균과 이동 표준편차 특징 추가
# 중요: groupby('unit_number')로 엔진별로 따로 계산해야 함!
df['sensor_3_roll_mean'] = df.groupby('unit_number')['sensor_3'].transform(
    lambda x: x.rolling(window_size, min_periods=1).mean()
)
df['sensor_3_roll_std'] = df.groupby('unit_number')['sensor_3'].transform(
    lambda x: x.rolling(window_size, min_periods=1).std()
)
```

transform의 역할

- 그룹별로 계산한 결과를, 원래 데이터프레임의 '모든 행'에 그대로 다시 붙여줌
- 그룹별 계산을 하지만 결과의 행 개수를 원본과 동일하게 유지시켜줌
- 이 덕분에 새로운 특성(열)으로 바로 추가할 수 있음

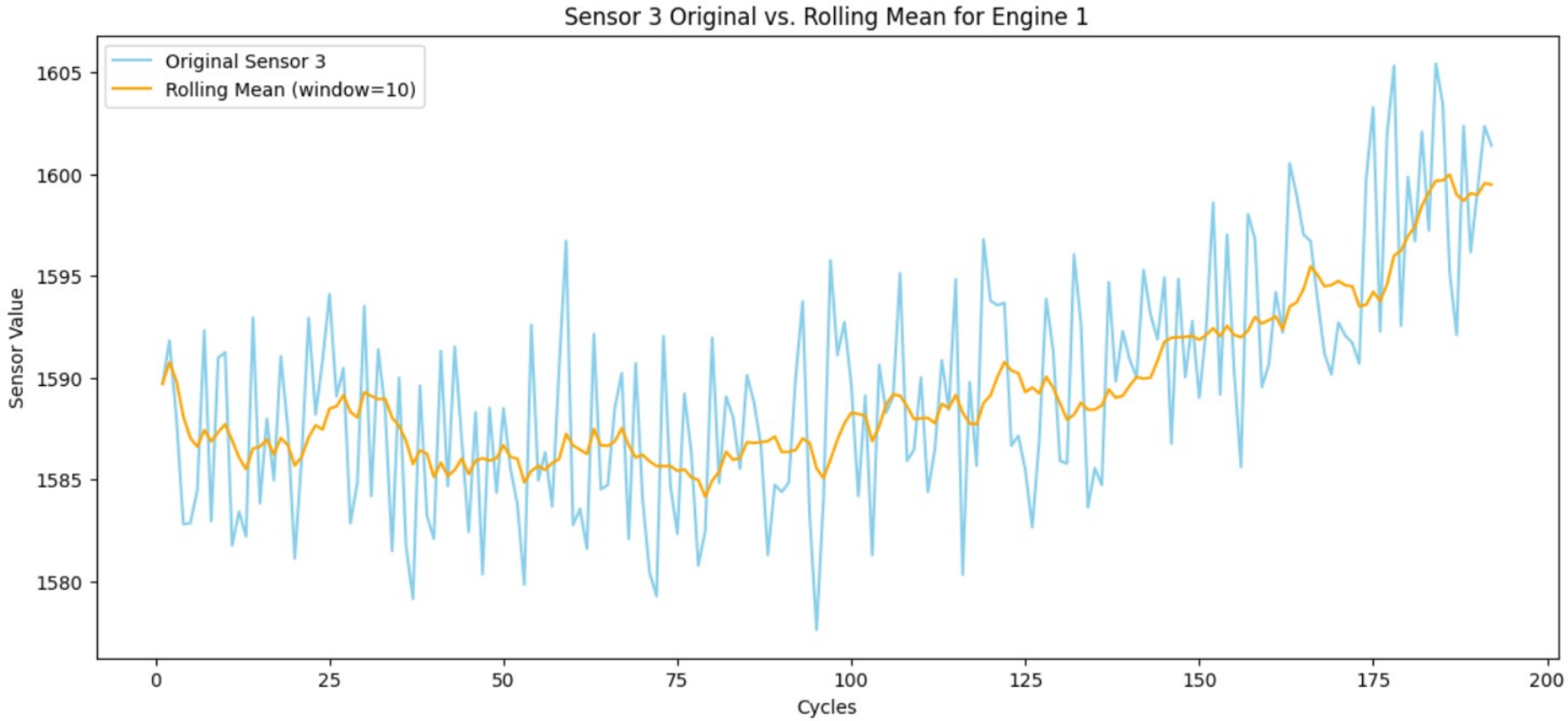
롤링 통계 실습: Pandas rolling() 함수 활용

- 결과 시각화: 이동 평균

```
# 4. 특정 엔진(예: 1번 엔진)의 데이터 시각화로 결과 확인
engine_1_df = df[df['unit_number'] == 1]

plt.figure(figsize=(14, 6))
plt.plot(engine_1_df['cycles'], engine_1_df['sensor_3'], label='Original Sensor 3', color='skyblue')
plt.plot(engine_1_df['cycles'], engine_1_df['sensor_3_roll_mean'],
         label=f'Rolling Mean (window={window_size})', color='orange')
plt.legend()
plt.title('Sensor 3 Original vs. Rolling Mean for Engine 1')
plt.xlabel('Cycles')
plt.ylabel('Sensor Value')
plt.show()
```

롤링 통계 실습: Pandas rolling() 함수 활용

- 결과 시각화: 이동 평균



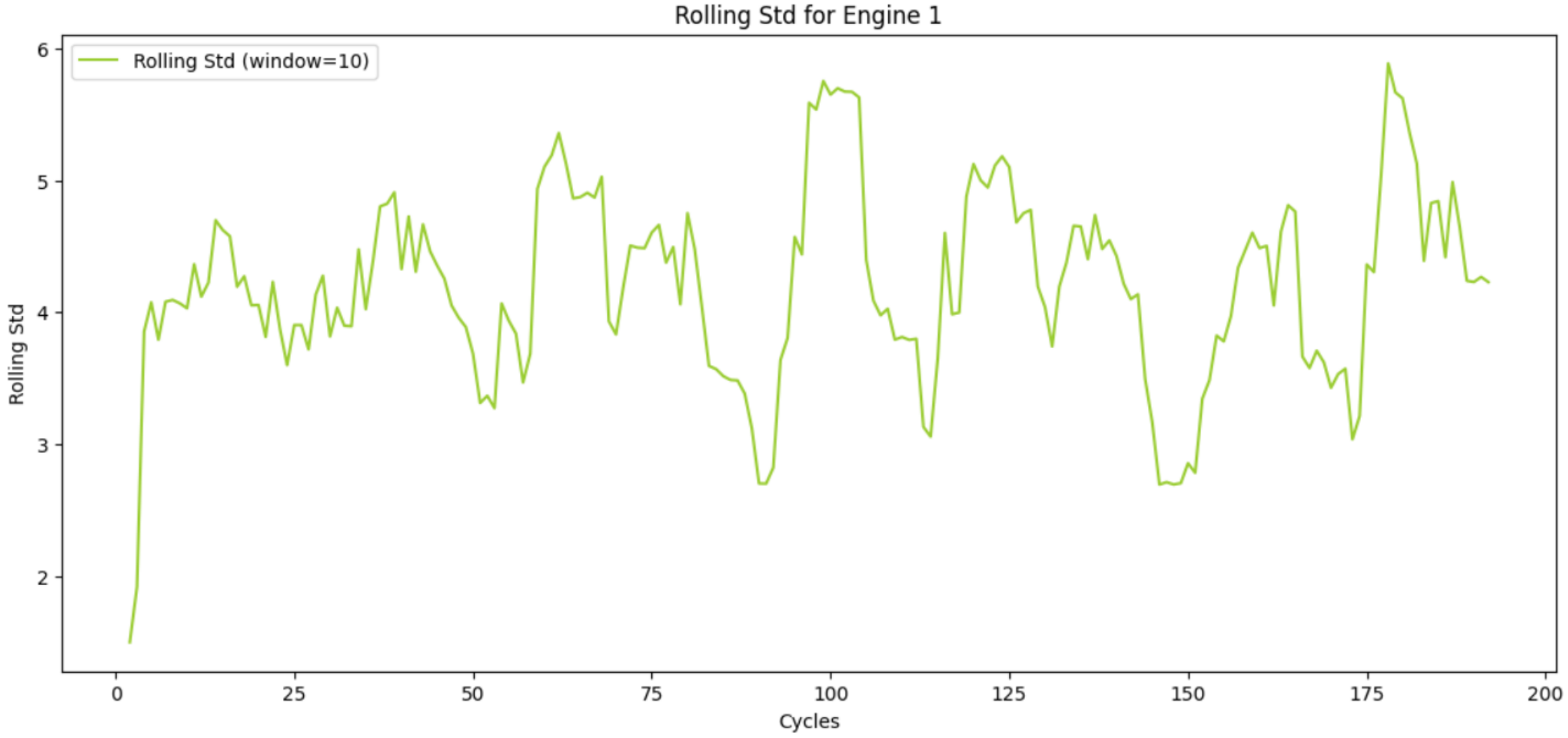
롤링 통계 실습: Pandas rolling() 함수 활용

- 결과 시각화 : 이동 편차

```
# 4. 특정 엔진(예: 1번 엔진)의 데이터 시각화로 결과 확인
engine_1_df = df[df['unit_number'] == 1]

plt.figure(figsize=(14, 6))
plt.plot(engine_1_df['cycles'], engine_1_df['sensor_3_roll_std'],
         label=f'Rolling Std (window={window_size})', color='yellowgreen')
plt.legend()
plt.title('Rolling Std for Engine 1')
plt.xlabel('Cycles')
plt.ylabel('Rolling Std')
plt.show()
```

롤링 통계 실습: Pandas rolling() 함수 활용

- 결과 시각화 : 이동 편차



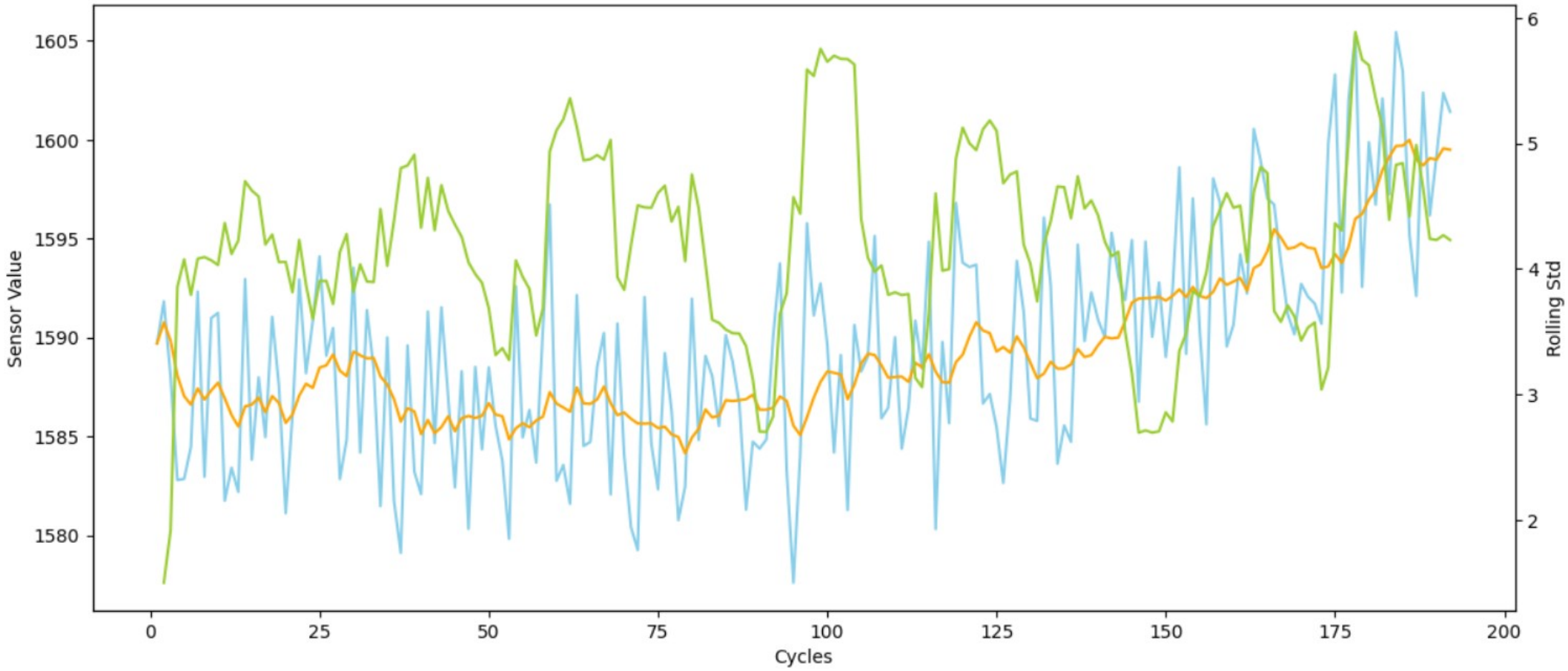
롤링 통계 실습: Pandas rolling() 함수 활용

- 결과 시각화 : 이동 평균과 이동 편차를 한 번에 그리기

```
# 4. 특정 엔진(예: 1번 엔진)의 데이터 시각화로 결과 확인
engine_1_df = df[df['unit_number'] == 1]

plt.rcParams['figure.figsize'] = (14, 6)

fig, ax1 = plt.subplots()
ax1.plot(engine_1_df['cycles'], engine_1_df['sensor_3'], label='Original Sensor 3', color='skyblue')
ax1.plot(engine_1_df['cycles'], engine_1_df['sensor_3_roll_mean'],
         label=f'Rolling Mean (window={window_size})', color='orange')
ax1.set_xlabel('Cycles')
ax1.set_ylabel('Sensor Value')

ax2 = ax1.twinx()
ax2.plot(engine_1_df['cycles'], engine_1_df['sensor_3_roll_std'],
         label=f'Rolling Std (window={window_size})', color='yellowgreen') #linestyle='--'
ax2.set_ylabel('Rolling Std')

plt.show()
```

롤링 통계 실습: Pandas rolling() 함수 활용

- 결과 시각화 : 이동 평균과 이동 편차를 한 번에 그리기



지연 특성과 누적 통계 : 과거와 현재를 연결하기

- 개념
  - 현재 시점의 데이터가 과거의 어떤 값에 영향을 받았는지 모델에게 알려주기 위해 생성하는 특성

- 지연 특징 (Lag Features)
  - 과거의 데이터를 현재로 가져옴
  - 1주기 전, 2주기 전의 센서 값을 현재 시점의 새로운 특징으로 추가

  - 왜 필요한가?
    - 머신러닝 모델은 기본적으로 각 데이터 행(row)을 독립적으로 봄
    - 지연 특징을 추가함으로써, 모델이 시간의 연속성("1주기 전 온도가 이랬기 때문에 현재 압력이 이렇다")을 학습하게 만들 수 있음

지연 특성과 누적 통계 : 과거와 현재를 연결하기

- 누적 통계 (Cumulative Statistics)
  - 데이터가 시작된 시점부터 현재까지의 통계를 누적하여 계산
  - 누적 합 (Cumulative Sum)
    - 장비가 가동된 이후 받은 총 스트레스 또는 총 작업량을 나타내는 지표로 활용될 수 있습니다.
  - 누적 평균 (Cumulative Mean)
    - 전체 기간 동안의 평균적인 상태를 나타냅니다.

실습: shift()와 cumsum() 함수 활용

- shift() 함수로 바로 이전 사이클의 센서 값을 가져오고, cumsum()으로 센서 값의 누적 합을 계산

```
# 1. 지연 특징 추가 (1 사이클 전 sensor_4 값)
# 중요: 역시 groupby()를 사용하여 엔진별로 계산해야 데이터가 섞이지 않음
df['sensor_4_lag_1'] = df.groupby('unit_number')['sensor_4'].transform(lambda x: x.shift(1))

# 2. 누적 합 특징 추가
df['sensor_7_cumsum'] = df.groupby('unit_number')['sensor_7'].transform(lambda x: x.cumsum())
```

실습: shift()와 cumsum() 함수 활용

```
# 3. 결과 확인 (결측치가 발생하는 첫 행 처리의 중요성도 함께 설명)
# shift()를 사용하면 첫 번째 행에 결측치(NaN)가 생기므로 처리 필요
print("Lag feature 생성 후 결측치 확인:")
df[['unit_number', 'cycles', 'sensor_4', 'sensor_4_lag_1']].head()
```

Lag feature 생성 후 결측치 확인:

	unit_number	cycles	sensor_4	sensor_4_lag_1
0	1	1	1400.60	NaN
1	1	2	1403.14	1400.60
2	1	3	1404.20	1403.14
3	1	4	1401.87	1404.20
4	1	5	1406.22	1401.87

```
# 결측치 채우기 (예: 0으로 채우거나, ffill)
df.fillna(0, inplace=True)
```

	unit_number	cycles	sensor_4	sensor_4_lag_1
0	1	1	1400.60	0.00
1	1	2	1403.14	1400.60
2	1	3	1404.20	1403.14
3	1	4	1401.87	1404.20
4	1	5	1406.22	1401.87

참고 자료

- 『제조AI빅데이터 분석 기법 (코딩편)』, 김일중 외 6인 공저, 보민출판사 (2024)
- 『혼자 공부하는 머신러닝 딥러닝』

끝!