10.11 단순 RNN과 LSTM, GPU 모델의 비교 - 시퀀스 데이터 준비

```
# 연속된 숫자 시퀀스 데이터와 레이블을 활용하여 순환 신경망의 모델이 잘 작동하는지 확인
import numpy as np

# 데이터를 생성하기 위한 sequence_gen() 함수 사용
# 0.0, 0.1, .. 증가하는 시퀀스 데이터를 생성
# seq_len 길이를 가지는 시퀀스 데이터를 size 갯수만큼 생성
def sequence_gen(size, seq_len):
  # 비어있는 넘파이 배열 생성
  seq_X = np.empty(shape=(size, seq_len, 1))
  Y = np.empty(shape=(size,))

  for i in range(size):
    # [0, 0.1, 0.2, .. ] 같은 시퀀스와 Y 값을 size 갯수만큼 생성
    c = np.linspace(i/10, (i+seq_len-1)/10, seq_len)
    # 새로운 축을 하나 더 추가
    seq_X[i] = c[:, np.newaxis]
    # 목표값 생성
    Y[i] = (i+seq_len) / 10

  return seq_X, Y

# 길이가 16인 시퀀스 8개를 훈련용으로 만든다
n, seq_len = 8, 16
train_seq_X, train_Y = sequence_gen(n, seq_len)
```

```
# 이전에 만든 훈련용 데이터를 flatten()함수를 활용하여 1줄씩 출력
print('훈련용 데이터')
for i in range(n):
  print(train_seq_X[i].flatten(), train_Y[i])

half_n, offset = int(n/2), 1.0
# 1.0만큼의 offset을 더해 테스트 셋 구성
test_seq_X = train_seq_X[:half_n] + offset
# 테스트 셋의 레이블 구성
test_Y = train_Y[:half_n] + offset

# 검증용 데이터도 비슷하게 출력
print('검증용 데이터')
for i in range(half_n):
  print(test_seq_X[i].flatten(), test_Y[i])
```

```
훈련용 데이터
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5] 1.6
[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6] 1.7
[0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7] 1.8
[0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8] 1.9
[0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9] 2.0
[0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ] 2.1
[0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1] 2.2
[0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2] 2.3
검증용 데이터
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5] 2.6
[1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6] 2.7
[1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7] 2.8
[1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8] 2.9
```

10.11 단순 RNN과 LSTM, GPU 모델의 비교 - 성능 비교

```
# SimpleRNN의 모델 성능
import tensorflow as tf
# 유닛의 개수 256개
n_units = 256
simpleRNN_model = tf.keras.Sequential([
    # 레이어를 구성하는 유닛개수 256개 지정
    tf.keras.layers.SimpleRNN(units = n_units, return_sequences=False,
                              input_shape=[seq_len, 1]),
    tf.keras.layers.Dense(1)
])

simpleRNN_model.compile(optimizer = 'adam', loss = 'mse')
# 100에폭으로 학습 진행
simpleRNN_model.fit(train_seq_X, train_Y, epochs = 100)
```

```
1/1 [==============================] - 0s 17ms/step - loss: 9.6452e-04
Epoch 73/100
1/1 [==============================] - 0s 15ms/step - loss: 9.6821e-04
Epoch 74/100
1/1 [==============================] - 0s 16ms/step - loss: 8.7347e-04
Epoch 75/100
1/1 [==============================] - 0s 21ms/step - loss: 7.7485e-04
Epoch 76/100
1/1 [==============================] - 0s 19ms/step - loss: 7.5563e-04
Epoch 77/100
1/1 [==============================] - 0s 16ms/step - loss: 8.0631e-04
Epoch 78/100
1/1 [==============================] - 0s 17ms/step - loss: 8.4990e-04
Epoch 79/100
1/1 [==============================] - 0s 17ms/step - loss: 8.3065e-04
Epoch 80/100
1/1 [==============================] - 0s 21ms/step - loss: 7.6677e-04
Epoch 81/100
1/1 [==============================] - 0s 20ms/step - loss: 7.1948e-04
Epoch 82/100
1/1 [==============================] - 0s 15ms/step - loss: 7.2291e-04
Epoch 83/100
1/1 [==============================] - 0s 17ms/step - loss: 7.5321e-04
Epoch 84/100
1/1 [==============================] - 0s 19ms/step - loss: 7.6293e-04
Epoch 85/100
1/1 [==============================] - 0s 16ms/step - loss: 7.3481e-04
Epoch 86/100
1/1 [==============================] - 0s 17ms/step - loss: 6.9503e-04
Epoch 87/100
1/1 [==============================] - 0s 17ms/step - loss: 6.7804e-04
Epoch 88/100
1/1 [==============================] - 0s 16ms/step - loss: 6.8819e-04
Epoch 89/100
1/1 [==============================] - 0s 16ms/step - loss: 7.0016e-04
Epoch 90/100
1/1 [==============================] - 0s 20ms/step - loss: 6.9123e-04
Epoch 91/100
1/1 [==============================] - 0s 18ms/step - loss: 6.6555e-04
Epoch 92/100
1/1 [==============================] - 0s 18ms/step - loss: 6.4509e-04
Epoch 93/100
1/1 [==============================] - 0s 16ms/step - loss: 6.4248e-04
Epoch 94/100
1/1 [==============================] - 0s 16ms/step - loss: 6.4856e-04
Epoch 95/100
1/1 [==============================] - 0s 18ms/step - loss: 6.4626e-04
Epoch 96/100
1/1 [==============================] - 0s 18ms/step - loss: 6.3113e-04
Epoch 97/100
1/1 [==============================] - 0s 17ms/step - loss: 6.1428e-04
Epoch 98/100
1/1 [==============================] - 0s 15ms/step - loss: 6.0707e-04
Epoch 99/100
1/1 [==============================] - 0s 16ms/step - loss: 6.0805e-04
Epoch 100/100
1/1 [==============================] - 0s 19ms/step - loss: 6.0672e-04
<keras.callbacks.History at 0x7f6373dfd130>
```

```
# 정답을 잘 예측하는지 확인
result = simpleRNN_model.predict(test_seq_X)
result = result.flatten()
print('예측값 :', result)
print('실제값 :', test_Y)
```

```
1/1 [==============================] - 0s 179ms/step
예측값 : [2.4263902 2.4715235 2.511855  2.5478733]
실제값 : [2.6 2.7 2.8 2.9]
```

```
# LSTM모델 성능
LSTM_model = tf.keras.Sequential([
```

```
    tf.keras.layers.LSTM(units = n_units, return_sequences=False,
                         input_shape=[seq_len, 1]),
    tf.keras.layers.Dense(1)
])

LSTM_model.compile(optimizer = 'adam', loss = 'mse')
LSTM_model.fit(train_seq_X, train_Y, epochs=100)
```

```
1/1 [==============================] - 0s 47ms/step - loss: 8.5778e-04
Epoch 73/100
1/1 [==============================] - 0s 45ms/step - loss: 9.1957e-04
Epoch 74/100
1/1 [==============================] - 0s 45ms/step - loss: 8.5671e-04
Epoch 75/100
1/1 [==============================] - 0s 41ms/step - loss: 7.1065e-04
Epoch 76/100
1/1 [==============================] - 0s 44ms/step - loss: 5.6578e-04
Epoch 77/100
1/1 [==============================] - 0s 48ms/step - loss: 4.9114e-04
Epoch 78/100
1/1 [==============================] - 0s 43ms/step - loss: 5.0260e-04
Epoch 79/100
1/1 [==============================] - 0s 53ms/step - loss: 5.6395e-04
Epoch 80/100
1/1 [==============================] - 0s 44ms/step - loss: 6.1787e-04
Epoch 81/100
1/1 [==============================] - 0s 42ms/step - loss: 6.2271e-04
Epoch 82/100
1/1 [==============================] - 0s 43ms/step - loss: 5.7341e-04
Epoch 83/100
1/1 [==============================] - 0s 39ms/step - loss: 4.9796e-04
Epoch 84/100
1/1 [==============================] - 0s 49ms/step - loss: 4.3565e-04
Epoch 85/100
1/1 [==============================] - 0s 44ms/step - loss: 4.1203e-04
Epoch 86/100
1/1 [==============================] - 0s 39ms/step - loss: 4.2561e-04
Epoch 87/100
1/1 [==============================] - 0s 36ms/step - loss: 4.5286e-04
Epoch 88/100
1/1 [==============================] - 0s 39ms/step - loss: 4.6612e-04
Epoch 89/100
1/1 [==============================] - 0s 42ms/step - loss: 4.5157e-04
Epoch 90/100
1/1 [==============================] - 0s 59ms/step - loss: 4.1570e-04
Epoch 91/100
1/1 [==============================] - 0s 44ms/step - loss: 3.7761e-04
Epoch 92/100
1/1 [==============================] - 0s 46ms/step - loss: 3.5466e-04
Epoch 93/100
1/1 [==============================] - 0s 43ms/step - loss: 3.5182e-04
Epoch 94/100
1/1 [==============================] - 0s 40ms/step - loss: 3.6075e-04
Epoch 95/100
1/1 [==============================] - 0s 49ms/step - loss: 3.6753e-04
Epoch 96/100
1/1 [==============================] - 0s 48ms/step - loss: 3.6236e-04
Epoch 97/100
1/1 [==============================] - 0s 40ms/step - loss: 3.4498e-04
Epoch 98/100
1/1 [==============================] - 0s 43ms/step - loss: 3.2314e-04
Epoch 99/100
1/1 [==============================] - 0s 37ms/step - loss: 3.0632e-04
Epoch 100/100
1/1 [==============================] - 0s 42ms/step - loss: 2.9933e-04
<keras.callbacks.History at 0x7f6373c9b160>
```

```
reulst = LSTM_model.predict(test_seq_X)
result = result.flatten()
print('예측값 :', result)
print('실제값', test_Y)
```

```
1/1 [==============================] - 0s 480ms/step
예측값 : [2.4263902 2.4715235 2.511855  2.5478733]
실제값 [2.6 2.7 2.8 2.9]
```

```
# GRU모델 성능
GRU_model = tf.keras.Sequential([
    tf.keras.layers.GRU(units = n_units, return_sequences=False,
                        input_shape=[seq_len, 1]),
    tf.keras.layers.Dense(1)
])

GRU_model.compile(optimizer = 'adam', loss = 'mse')
GRU_model.fit(train_seq_X, train_Y, epochs=100)
```

```
1/1 [==============================] - 0s 41ms/step - loss: 4.1450e-04
Epoch 74/100
1/1 [==============================] - 0s 33ms/step - loss: 4.3785e-04
Epoch 75/100
1/1 [==============================] - 0s 43ms/step - loss: 4.0662e-04
Epoch 76/100
1/1 [==============================] - 0s 33ms/step - loss: 3.3313e-04
Epoch 77/100
1/1 [==============================] - 0s 34ms/step - loss: 2.3826e-04
Epoch 78/100
1/1 [==============================] - 0s 37ms/step - loss: 1.4509e-04
Epoch 79/100
1/1 [==============================] - 0s 36ms/step - loss: 7.2882e-05
Epoch 80/100
1/1 [==============================] - 0s 38ms/step - loss: 3.2913e-05
Epoch 81/100
1/1 [==============================] - 0s 34ms/step - loss: 2.6790e-05
Epoch 82/100
1/1 [==============================] - 0s 43ms/step - loss: 4.7477e-05
Epoch 83/100
1/1 [==============================] - 0s 35ms/step - loss: 8.2372e-05
Epoch 84/100
1/1 [==============================] - 0s 36ms/step - loss: 1.1745e-04
Epoch 85/100
1/1 [==============================] - 0s 38ms/step - loss: 1.4119e-04
Epoch 86/100
1/1 [==============================] - 0s 37ms/step - loss: 1.4728e-04
Epoch 87/100
1/1 [==============================] - 0s 36ms/step - loss: 1.3542e-04
Epoch 88/100
1/1 [==============================] - 0s 38ms/step - loss: 1.1044e-04
Epoch 89/100
1/1 [==============================] - 0s 35ms/step - loss: 8.0056e-05
Epoch 90/100
1/1 [==============================] - 0s 42ms/step - loss: 5.2233e-05
Epoch 91/100
1/1 [==============================] - 0s 41ms/step - loss: 3.2935e-05
Epoch 92/100
1/1 [==============================] - 0s 47ms/step - loss: 2.4830e-05
Epoch 93/100
1/1 [==============================] - 0s 33ms/step - loss: 2.7123e-05
Epoch 94/100
1/1 [==============================] - 0s 37ms/step - loss: 3.6385e-05
Epoch 95/100
1/1 [==============================] - 0s 37ms/step - loss: 4.7963e-05
Epoch 96/100
1/1 [==============================] - 0s 45ms/step - loss: 5.7468e-05
Epoch 97/100
1/1 [==============================] - 0s 42ms/step - loss: 6.1944e-05
Epoch 98/100
1/1 [==============================] - 0s 43ms/step - loss: 6.0415e-05
Epoch 99/100
1/1 [==============================] - 0s 37ms/step - loss: 5.3827e-05
Epoch 100/100
1/1 [==============================] - 0s 33ms/step - loss: 4.4436e-05
<keras.callbacks.History at 0x7f637375b490>
```

```
result = GRU_model.predict(test_seq_X)
result = result.flatten()
print('예측값 :', result)
print('실제값 :', test_Y)
```

```
1/1 [==============================] - 0s 403ms/step
예측값 : [2.5716743 2.6650834 2.7578063 2.8498316]
실제값 : [2.6 2.7 2.8 2.9]
```

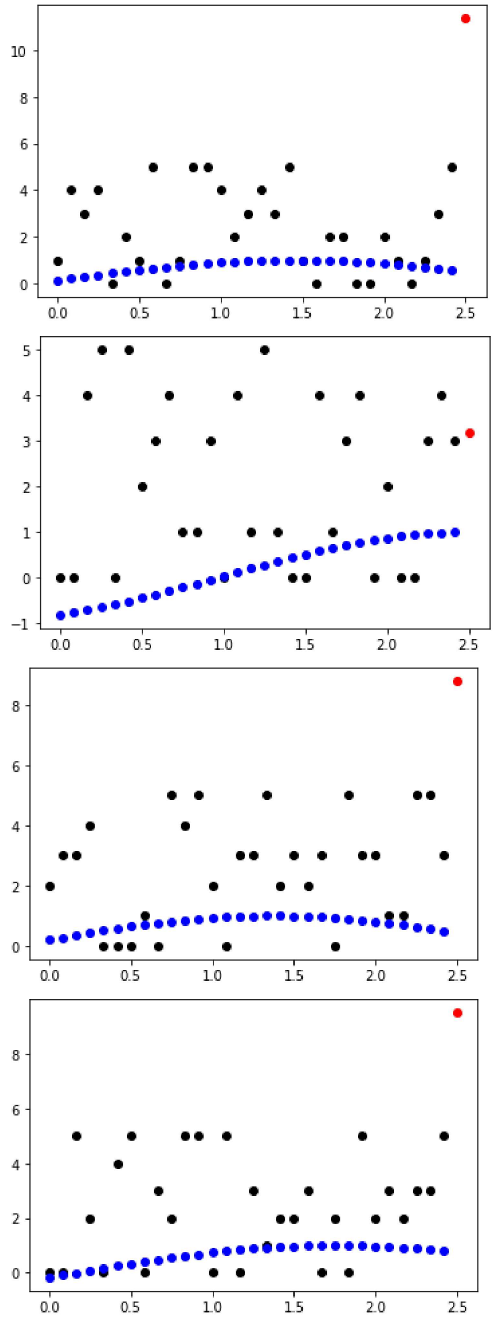
LAB 10-2 기억이 필요한 시퀀스 예측

실습 목표

> - 사인 곡선에서 일부분을 잘라 만든 시퀀스의 각 요소 각각에 임의의 난수 인덱스를 부여하자. 이번에는 이 시퀀스의 다음 값을 예측하는 것이 아니라, 시퀀스의 각 요소들 가운데 짝수 인덱스를 가진 요소들의 평균 값을 계산하는 모델을 만들어 보자.

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

# 시퀀스의 개수 200개 시퀀스의 길이 30으로 설정
size, seq_len = 200, 30
# 비어있는 넘파이 배열 생성
# 각 시퀀스에 인덱스가 존재
seq_X = np.empty(shape=(size, seq_len, 2))
```

```
# 각 시퀀스의 정답을 담은 변수생성
Y = np.empty(shape=(size,))

# sine 곡선에서 잘라낼 구간 설정
interval = np.linspace(0.0, 2.5, seq_len+1)


shift = np.random.randn(size)
# 시퀀스 내의 각 원소에 대해 인덱스와 값을 설정
for i in range(size):
  # 인덱스
  seq_X[i,:,0] = np.random.randint(0, 6, size=(seq_len))
  # 값
  seq_X[i,:,1] = np.sin(shift[i] + interval[:-1])
  # 정답 레이블은 시퀀스 내에서 짝수 인덱스를 가진 원소의 값을 모두 더한 값
  even_idx = seq_X[i, seq_X[i,:,0]%2 == 0]
  Y[i] = even_idx[:,1].sum()


for i in [1, 3, 5, 9]:
  # 인덱스 정보
  plt.scatter(interval[:-1], seq_X[i, :, 0], color='k')
  # 값: 사인 시퀀스 파란색 선으로 나타난 점으로 표시
  plt.scatter(interval[:-1], seq_X[i, :, 1], color='b')
  # 레이블 붉은 점으로 표시
  plt.scatter(interval[-1], Y[i], color='r')
  plt.show()
```



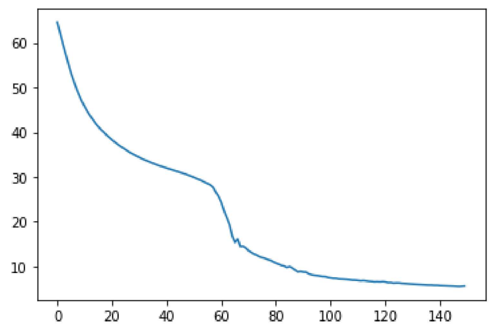
```
# 훈련용과 테스트용으로 나누기
train_X = seq_X[:180]
train_y = Y[:180]
test_X = seq_X[180:]
test_y = Y[180:]
```

```
import tensorflow as tf
simpleRNN_model = tf.keras.Sequential([
    # RNN 유닛의 수를 앞의 예제와 같이 10개로 설정, 20개의 연결
    tf.keras.layers.SimpleRNN(units = 10, return_sequences=False,
                              input_shape = [seq_len, 2]),
    tf.keras.layers.Dense(1)
])

simpleRNN_model.compile(optimizer = 'adam', loss = 'mse')
simpleRNN_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 10)                130

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0
_________________________________________________________________
```

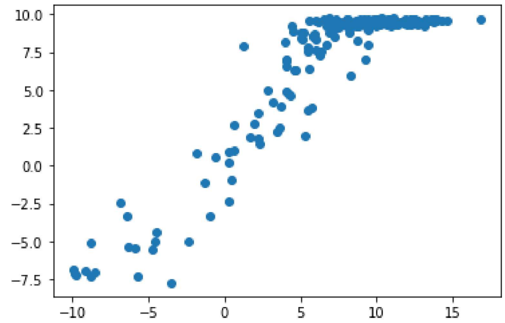
```
history = simpleRNN_model.fit(train_X, train_y, epochs=150)
plt.plot(history.history['loss'])
```

```
Epoch 131/150
6/6 [==============================] - 0s 8ms/step - loss: 5.9861
Epoch 132/150
6/6 [==============================] - 0s 6ms/step - loss: 5.9519
Epoch 133/150
6/6 [==============================] - 0s 7ms/step - loss: 5.9126
Epoch 134/150
6/6 [==============================] - 0s 6ms/step - loss: 5.8933
Epoch 135/150
6/6 [==============================] - 0s 6ms/step - loss: 5.8532
Epoch 136/150
6/6 [==============================] - 0s 6ms/step - loss: 5.8194
Epoch 137/150
6/6 [==============================] - 0s 6ms/step - loss: 5.7865
Epoch 138/150
6/6 [==============================] - 0s 6ms/step - loss: 5.7860
Epoch 139/150
6/6 [==============================] - 0s 7ms/step - loss: 5.7299
Epoch 140/150
6/6 [==============================] - 0s 7ms/step - loss: 5.7365
Epoch 141/150
6/6 [==============================] - 0s 6ms/step - loss: 5.6757
Epoch 142/150
6/6 [==============================] - 0s 9ms/step - loss: 5.6572
Epoch 143/150
6/6 [==============================] - 0s 6ms/step - loss: 5.6168
Epoch 144/150
6/6 [==============================] - 0s 9ms/step - loss: 5.5910
Epoch 145/150
6/6 [==============================] - 0s 8ms/step - loss: 5.5756
Epoch 146/150
6/6 [==============================] - 0s 6ms/step - loss: 5.5518
Epoch 147/150
6/6 [==============================] - 0s 6ms/step - loss: 5.5214
Epoch 148/150
6/6 [==============================] - 0s 6ms/step - loss: 5.4768
Epoch 149/150
6/6 [==============================] - 0s 6ms/step - loss: 5.5125
Epoch 150/150
6/6 [==============================] - 0s 6ms/step - loss: 5.5706
[<matplotlib.lines.Line2D at 0x7f636daa5460>]
```

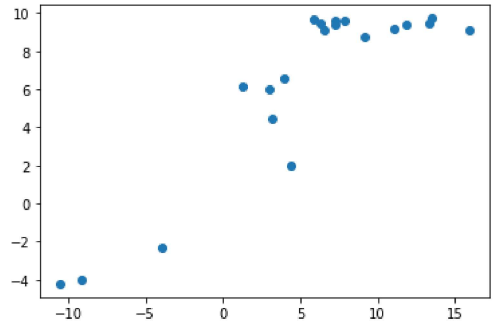


```
# 훈련 데이터에 대한 예측 결과와 실제 정답을 비교
train_y_hat = simpleRNN_model.predict(train_X)
plt.scatter(train_y, train_y_hat)
plt.show()
test_y_hat = simpleRNN_model.predict(test_X)
plt.scatter(test_y, test_y_hat)
plt.show()
```

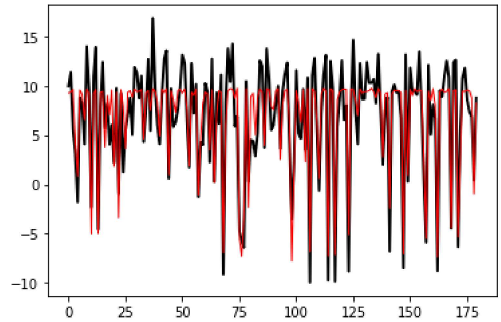
```
6/6 [==============================] - 0s 3ms/step
```



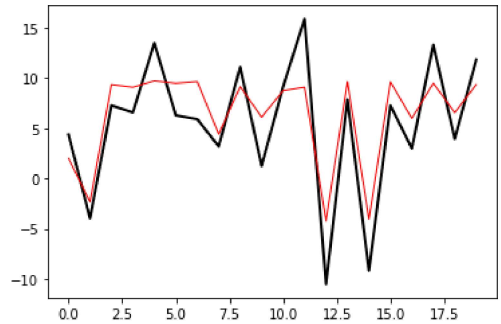
```
1/1 [==============================] - 0s 19ms/step
```



```
# 훈련용 데이터의 정답과 예측 값, 검증용 데이터의 정답과 예측값을 데이터에 들어있는 순서대고 그리기
# 정답값 검정색
plt.plot(train_y, c='k', linewidth=2)
# 예측값 빨간색
plt.plot(train_y_hat, c='r', linewidth=1)
plt.show()
# 정답값 검정색
plt.plot(test_y, c='k', linewidth=2)
# 예측값 빨간색
plt.plot(test_y_hat, c='r', linewidth=1)
```



```
[<matplotlib.lines.Line2D at 0x7f6373cddd00>]
```



```
LSTM_model = tf.keras.Sequential([
    tf.keras.layers.LSTM(units = 10, return_sequences=False,
                         input_shape=[seq_len, 2]),
```

```
    tf.keras.layers.Dense(1)
])

LSTM_model.compile(optimizer = 'adam', loss = 'mse')
LSTM_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 10)                520

 dense_4 (Dense)             (None, 1)                 11

=================================================================
Total params: 531
Trainable params: 531
Non-trainable params: 0
_________________________________________________________________
```

```
# 훈련과 같은 방식으로 에폭의 수 150개로 지정
history = LSTM_model.fit(train_X, train_y, epochs=150)
plt.plot(history.history['loss'])
```

```
Epoch 104/150
6/6 [==============================] - 0s 13ms/step - loss: 9.0412
Epoch 105/150
6/6 [==============================] - 0s 14ms/step - loss: 8.9338
Epoch 106/150
6/6 [==============================] - 0s 12ms/step - loss: 8.8300
Epoch 107/150
6/6 [==============================] - 0s 13ms/step - loss: 8.7289
Epoch 108/150
6/6 [==============================] - 0s 12ms/step - loss: 8.6301
Epoch 109/150
6/6 [==============================] - 0s 13ms/step - loss: 8.5226
Epoch 110/150
6/6 [==============================] - 0s 14ms/step - loss: 8.4396
Epoch 111/150
6/6 [==============================] - 0s 13ms/step - loss: 8.3383
Epoch 112/150
6/6 [==============================] - 0s 16ms/step - loss: 8.2354
Epoch 113/150
6/6 [==============================] - 0s 12ms/step - loss: 8.1538
Epoch 114/150
6/6 [==============================] - 0s 12ms/step - loss: 8.0826
Epoch 115/150
6/6 [==============================] - 0s 11ms/step - loss: 7.9963
Epoch 116/150
6/6 [==============================] - 0s 12ms/step - loss: 7.8777
Epoch 117/150
6/6 [==============================] - 0s 12ms/step - loss: 7.8177
Epoch 118/150
6/6 [==============================] - 0s 12ms/step - loss: 7.7439
Epoch 119/150
6/6 [==============================] - 0s 14ms/step - loss: 7.6574
Epoch 120/150
6/6 [==============================] - 0s 12ms/step - loss: 7.5712
Epoch 121/150
6/6 [==============================] - 0s 13ms/step - loss: 7.4935
Epoch 122/150
6/6 [==============================] - 0s 15ms/step - loss: 7.4275
Epoch 123/150
6/6 [==============================] - 0s 14ms/step - loss: 7.3493
Epoch 124/150
6/6 [==============================] - 0s 13ms/step - loss: 7.2728
Epoch 125/150
6/6 [==============================] - 0s 13ms/step - loss: 7.2000
Epoch 126/150
6/6 [==============================] - 0s 15ms/step - loss: 7.1401
Epoch 127/150
6/6 [==============================] - 0s 15ms/step - loss: 7.0620
Epoch 128/150
6/6 [==============================] - 0s 15ms/step - loss: 7.0402
Epoch 129/150
6/6 [==============================] - 0s 14ms/step - loss: 6.9273
Epoch 130/150
6/6 [==============================] - 0s 15ms/step - loss: 6.8986
Epoch 131/150
```