KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2022

스마트교통 빅데이터 분석

RNN을 이용한 시계열 데이터 예측하기

2022.8.30.

이 홍 석 (hsyi@kisti.re.kr)

시계열 예측하기

❖ 시계열

✓모든 데이터는 타임 스텝 마다 1개 이상의 값을 갖는다.

- 단별량 시계열은 타임 스텝 마다 1개의 값
- 다변량 시계열은 타임 스텝 마다 여러 개의 값

✓예측(forecasting)은 미래의 값을 예측하는 것

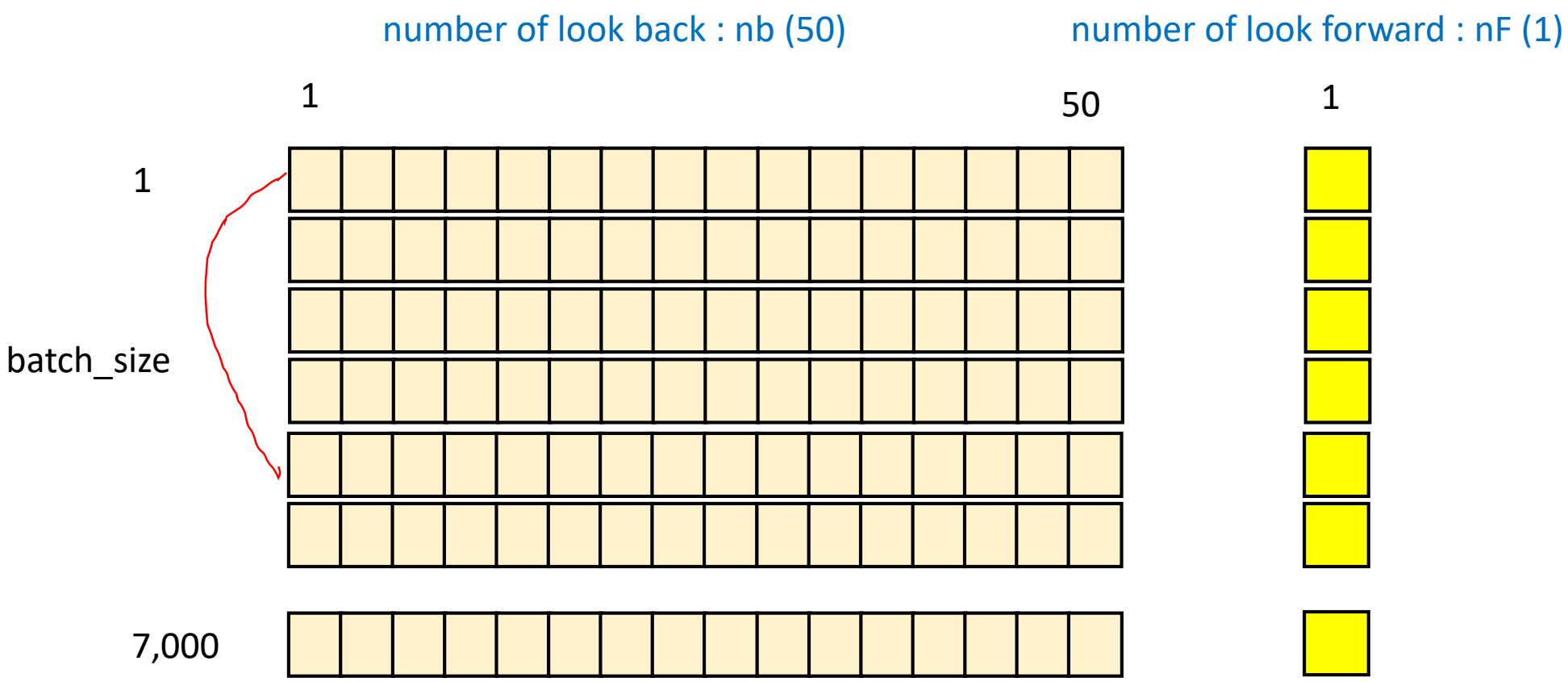
✓값 대체(imputation) 누락된 값을 예측하는 것

❖ 실습 문제

✓단별량 시계열 데이터의 타임 스텝은 50일때 각 위치에서 다음 타임의 값을 예측하라. (그램의 X표시)

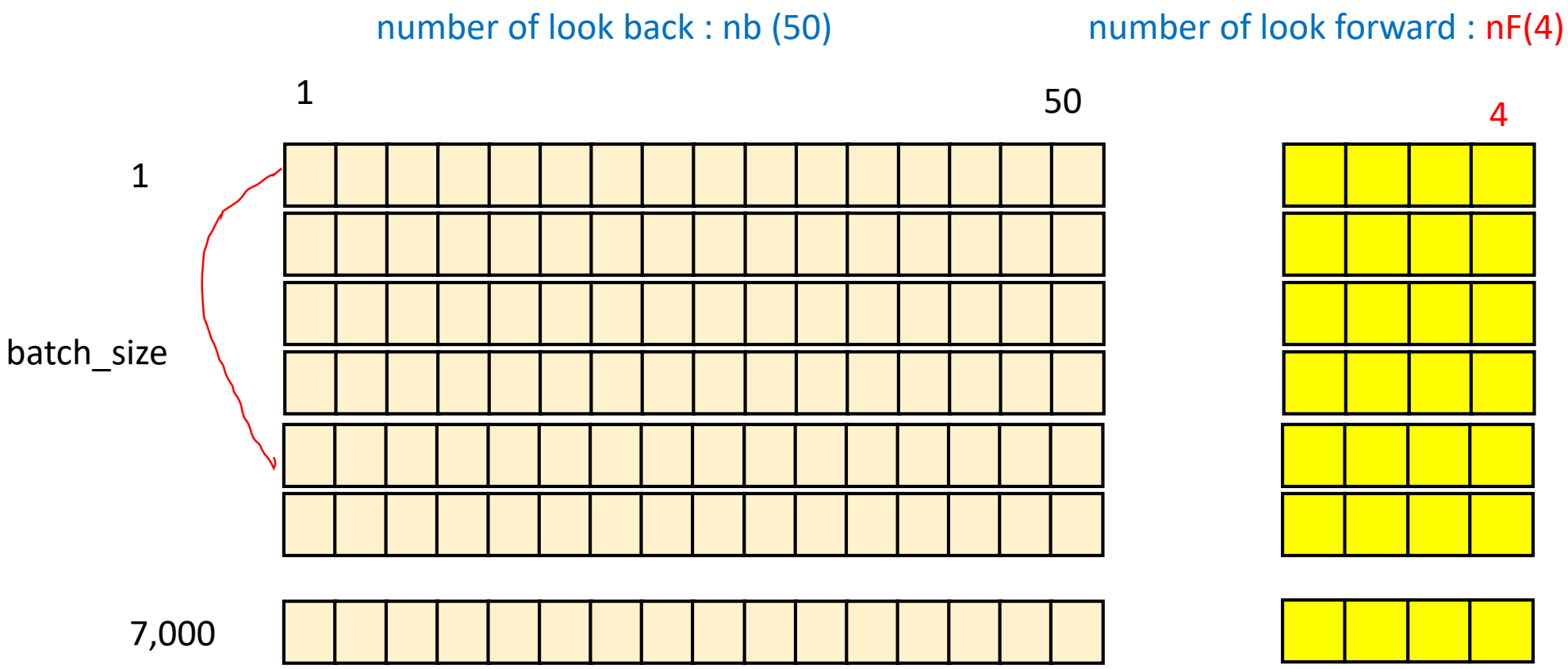
Many-to-One RNN Data Structure

X_train[7000,50,1] y_train[7000,1,1]

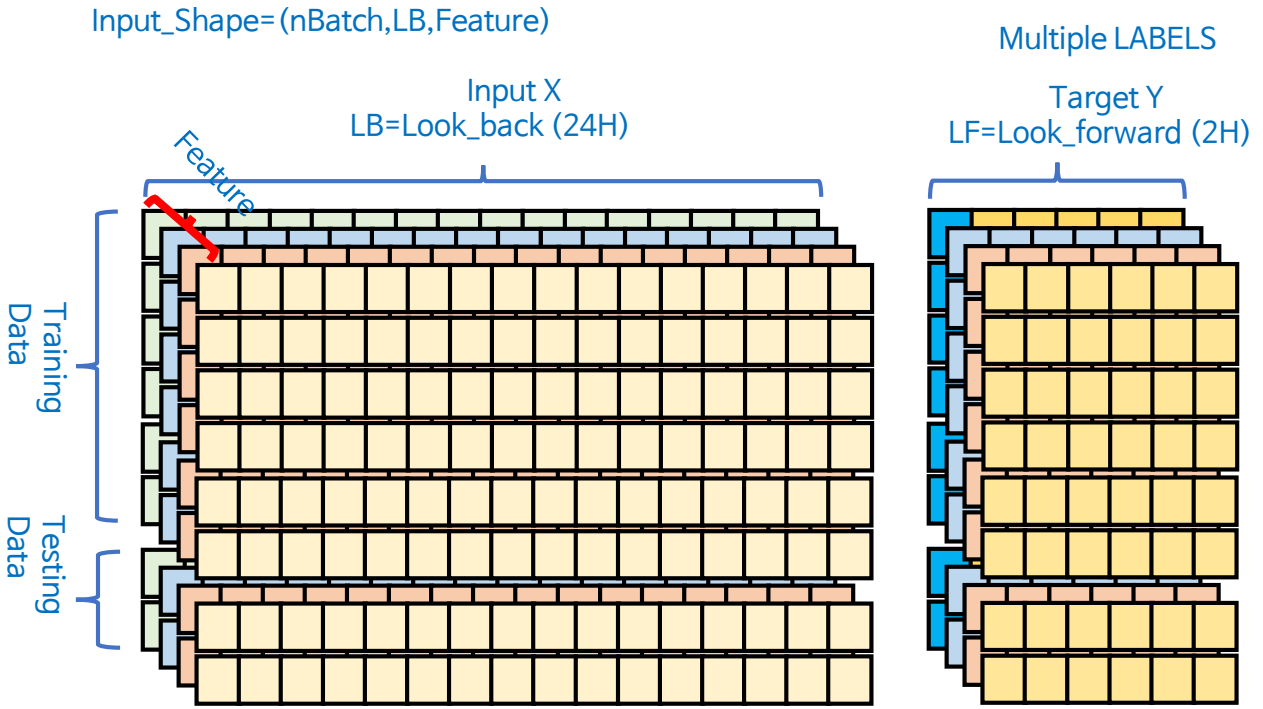


Many-to-Many RNN Data Structure

X_train[7000,50,1] y_train[7000,nF,1]



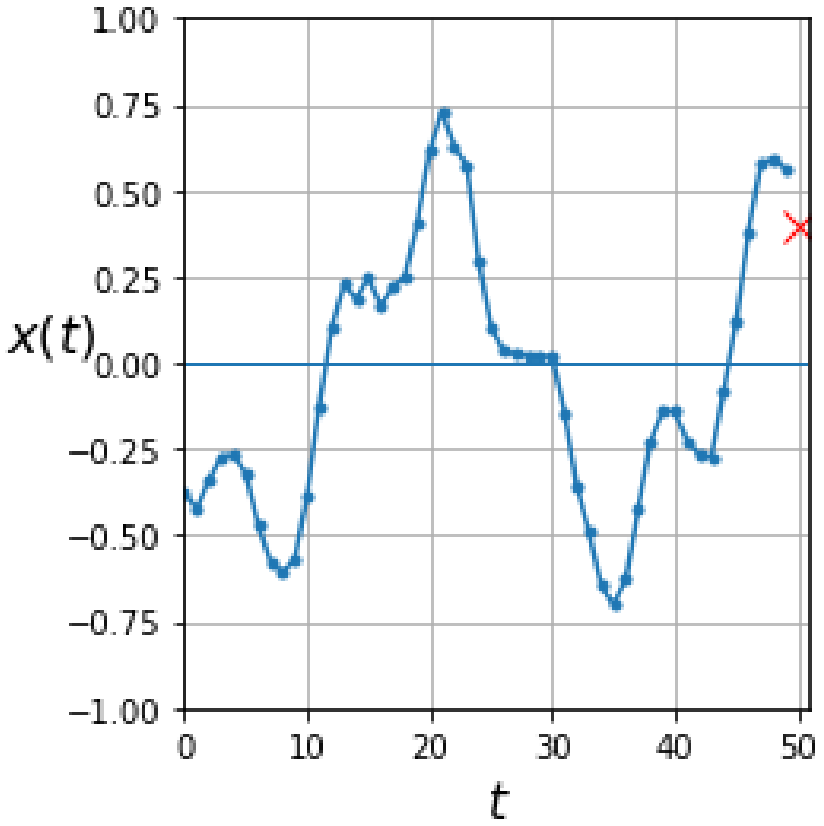
RNN Input-Output Data Structure



VDS, RSE Data Preprocessing for LSTM

단별량 시계열 데이터 생성

- sin() 함수 2개를 변형하여 합하고 일부 노이즈를 넣은 데이터
  - ✓배치사이즈: 10,000
  - ✓타임 스텝 : 50
  - ✓차원(값) : 1개



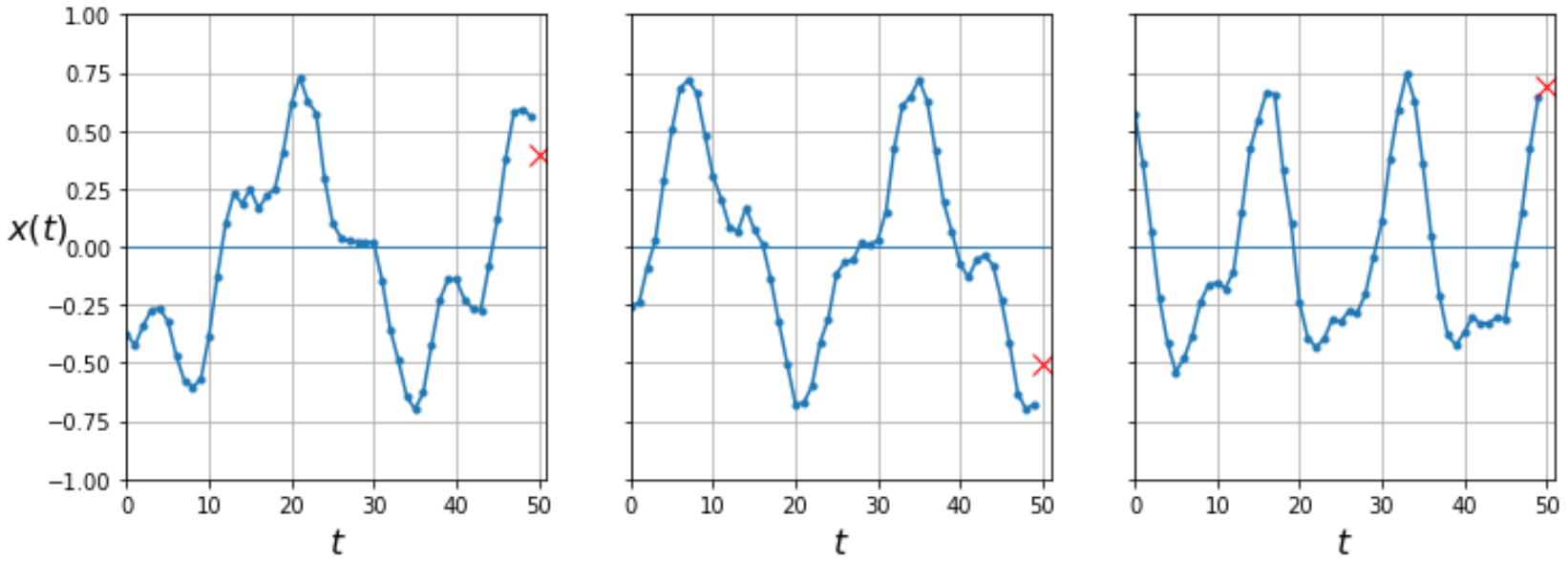
01: 데이터 생성

```
def generate_time_series(batch_size, n_steps):
    freq1, freq2, offsets1, offsets2 = np.random.rand(4, batch_size, 1)

    time = np.linspace(0, 1, n_steps)

    series = 0.5 * np.sin((time - offsets1) * (freq1 * 10 + 10))  #   w
    series += 0.2 * np.sin((time - offsets2) * (freq2 * 20 + 20)) # + w

    series += 0.1 * (np.random.rand(batch_size, n_steps) - 0.5)   # + n

    return series[..., np.newaxis].astype(np.float32)
```

단별량 시계열 데이터 생성

❖ 시계열 데이터 생성하는 일반적인 방법은 3D 배열로

데이터 A = [배치크기, 타입 스텝 수, 차원수]

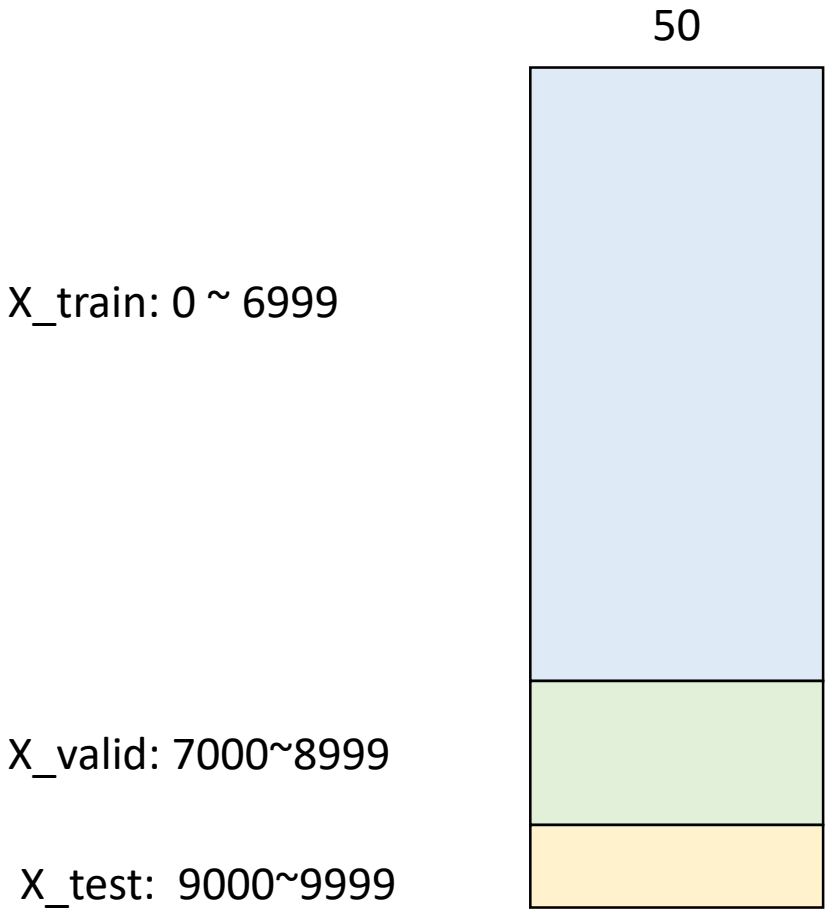


데이터 분할

```
np.random.seed(42)

n_steps = 50

series = generate_time_series(10000, n_steps + 1)


X_train, y_train = series[:7000, :n_steps], series[:7000, -1]

X_valid, y_valid = series[7000:9000, :n_steps], series[7000:9000, -1]

X_test, y_test = series[9000:, :n_steps], series[9000:, -1]


X_train.shape, y_train.shape, X_valid.shape
```

```
((7000, 50, 1), (7000, 1), (2000, 50, 1))
```

데이터 x_train



단변량 시계열 데이터 생성

```
def plot_series(series, y=None, y_pred=None, x_label="$t$", y_label="$x(t)$"):
    plt.plot(series, ".-")
    if y is not None:
        plt.plot(n_steps, y, "rx", markersize=10)
    if y_pred is not None:
        plt.plot(n_steps, y_pred, "ro")
    plt.grid(True)
    if x_label:
        plt.xlabel(x_label, fontsize=16)
    if y_label:
        plt.ylabel(y_label, fontsize=16, rotation=0)
    plt.hlines(0, 0, 100, linewidth=1)
    plt.axis([0, n_steps + 1, -1, 1])

fig, axes = plt.subplots(nrows=1, ncols=3, sharey=True, figsize=(12, 4))
for col in range(3):
    plt.sca(axes[col])
    plot_series(X_valid[col, :, 0], y_valid[col, 0],
                y_label=("$x(t)$" if col==0 else None))

plt.show()
```

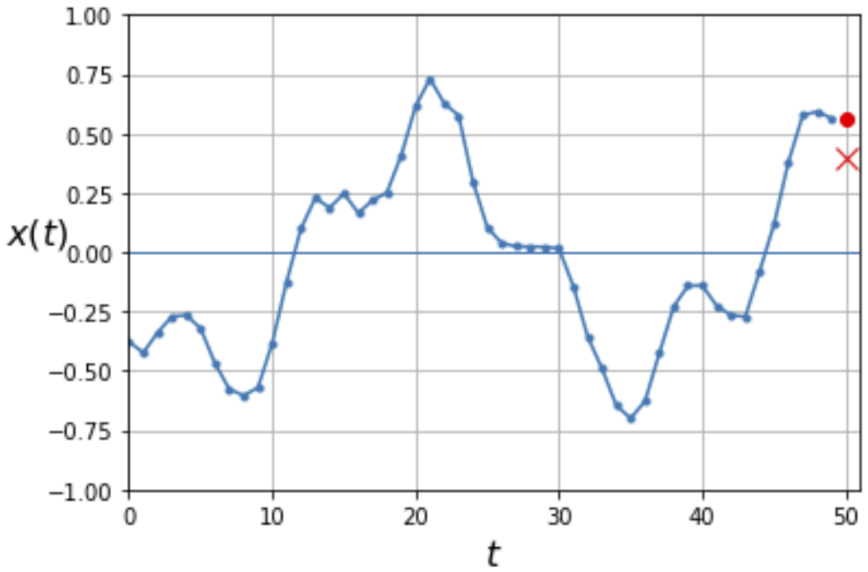
1. 기준 성능 정하자

❖ 순진한 예측 (native forecasting) : 시계열의 값을 그대로 예측해보자

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

```
0.020211367
```

```
plot_series(X_valid[0, :, 0], y_valid[0, 0], y_pred[0, 0])
plt.show()
```

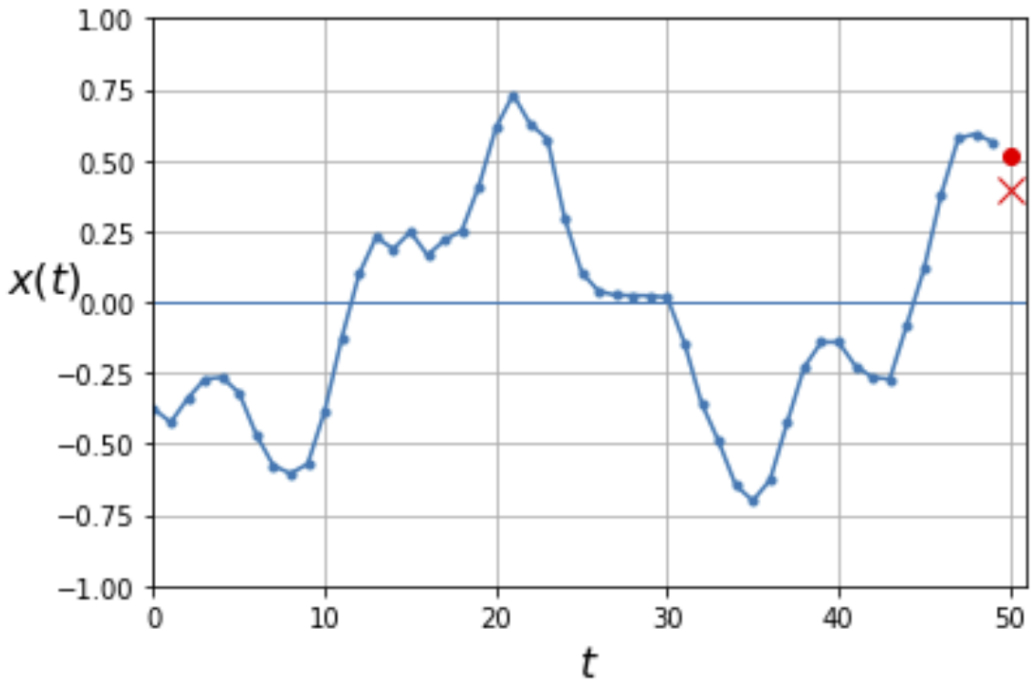


Loss=0.0202

- 이 신경망은 입력마다 1차원 특성을 기대하기 때문에 Flatten 층 사용함
- 시계열을 선형모델로 예측

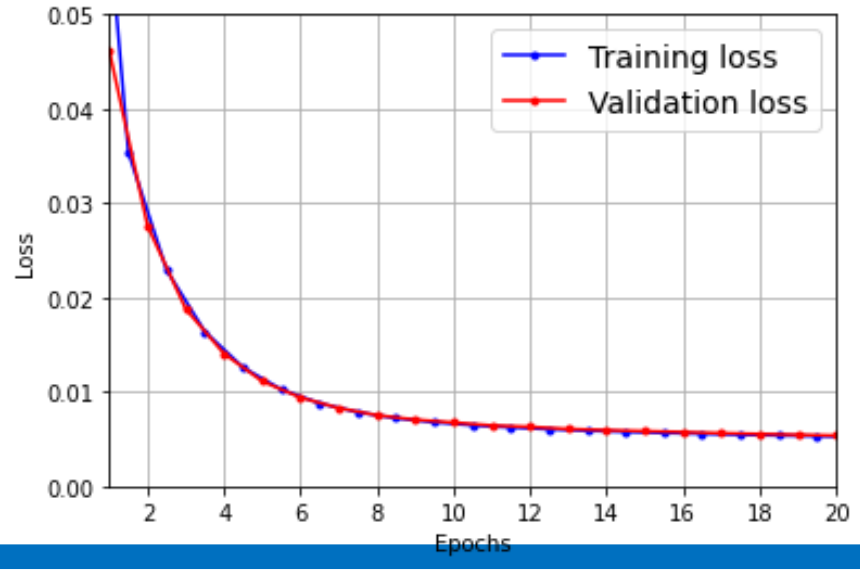
```
np.random.seed(42)
tf.random.set_seed(42)

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

model.compile(loss="mse", optimizer="sgd")
history = model.fit(X_train, y_train, epochs=100,
                    validation_data=(X_valid, y_valid))

model.evaluate(X_valid, y_valid)
```

```
y_pred = model.predict(X_valid)
plot_series(X_valid[0, :, 0], y_valid[0, 0], y_pred[0, 0])
plt.show()
```



Loss=0.0048

```python
def plot_learning_curves(loss, val_loss):
    plt.plot(np.arange(len(loss)) + 0.5, loss, "b.-", label="Training loss")
    plt.plot(np.arange(len(val_loss)) + 1, val_loss, "r.-", label="Validation loss"
    plt.gca().xaxis.set_major_locator(mpl.ticker.MaxNLocator(integer=True))
    plt.axis([1, 20, 0, 0.05])
    plt.legend(fontsize=14)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.grid(True)

plot_learning_curves(history.history["loss"], history.history["val_loss"])
plt.show()
```

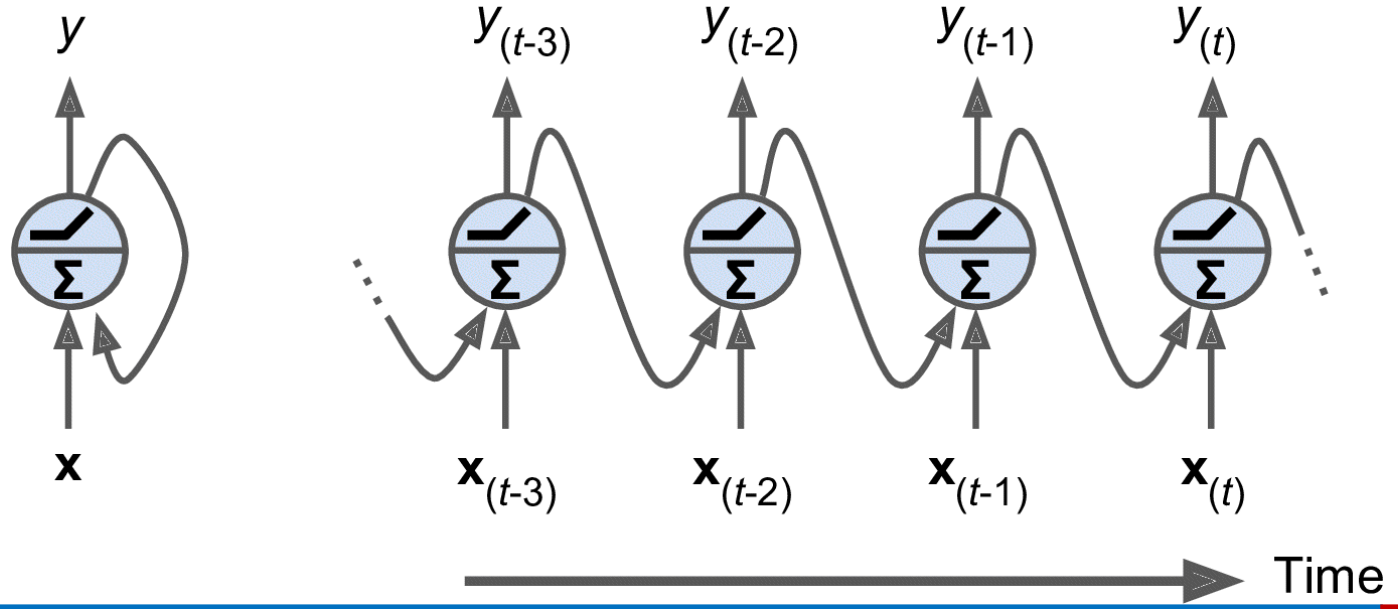


03. 간단한 RNN 모델

```
np.random.seed(42)
tf.random.set_seed(42)

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

optimizer = keras.optimizers.Adam(lr=0.005)
model.compile(loss="mse", optimizer=optimizer)
history = model.fit(X_train, y_train, epochs=100,
                    validation_data=(X_valid, y_valid))


model.evaluate(X_valid, y_valid)
```

❖ RNN 입력 시퀀스의 길이를 지정할 필요 없어서 입력에 None을 쓴다

- ✓SimpleRNN 층은 하이퍼볼릭 탄젠트 활성함수를 사용한다. (-1 ~ 1)
- ✓초기 상태 h(init)=0으로 설정하고 x(t=0)와 함께 순환 뉴런에 전달한다.
- ✓활성함수르 통해서 y(0)를 출력한다.
- ✓이 새로운 h(0)가 되며, 다음 입력 x(1)과 입력으로 전달.
- ✓마지막 층은 y(49)가 된다.



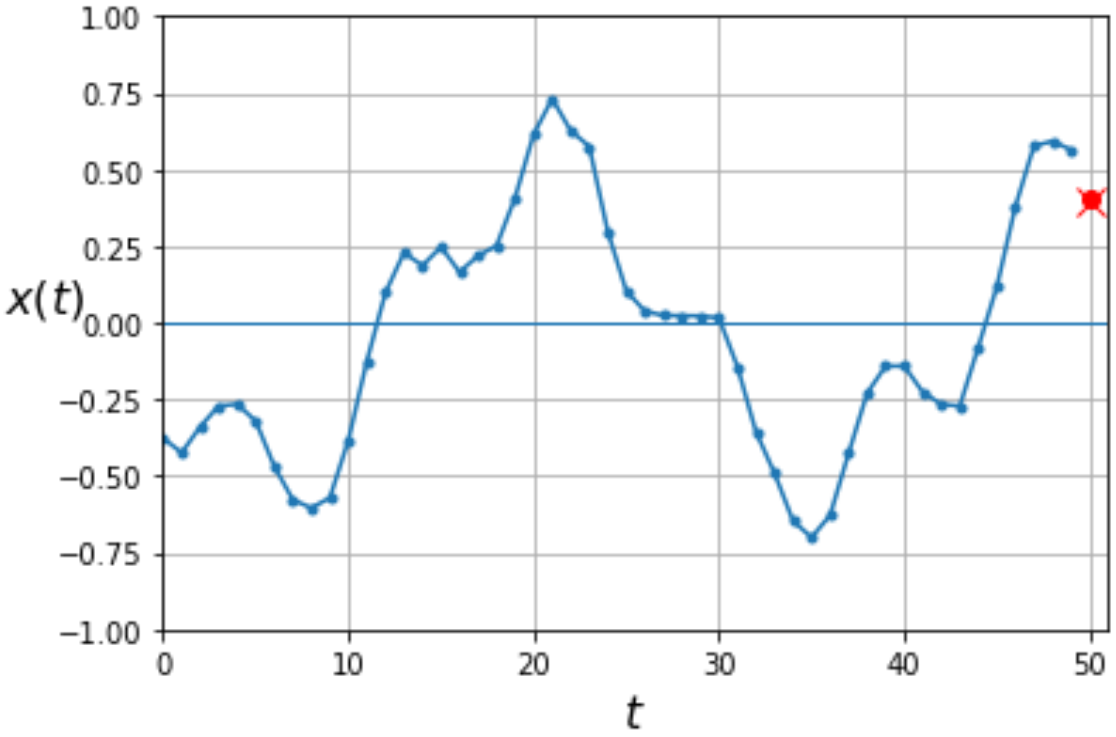
❖ SimpleRNN 결과와 선형모델과 비교해보세요

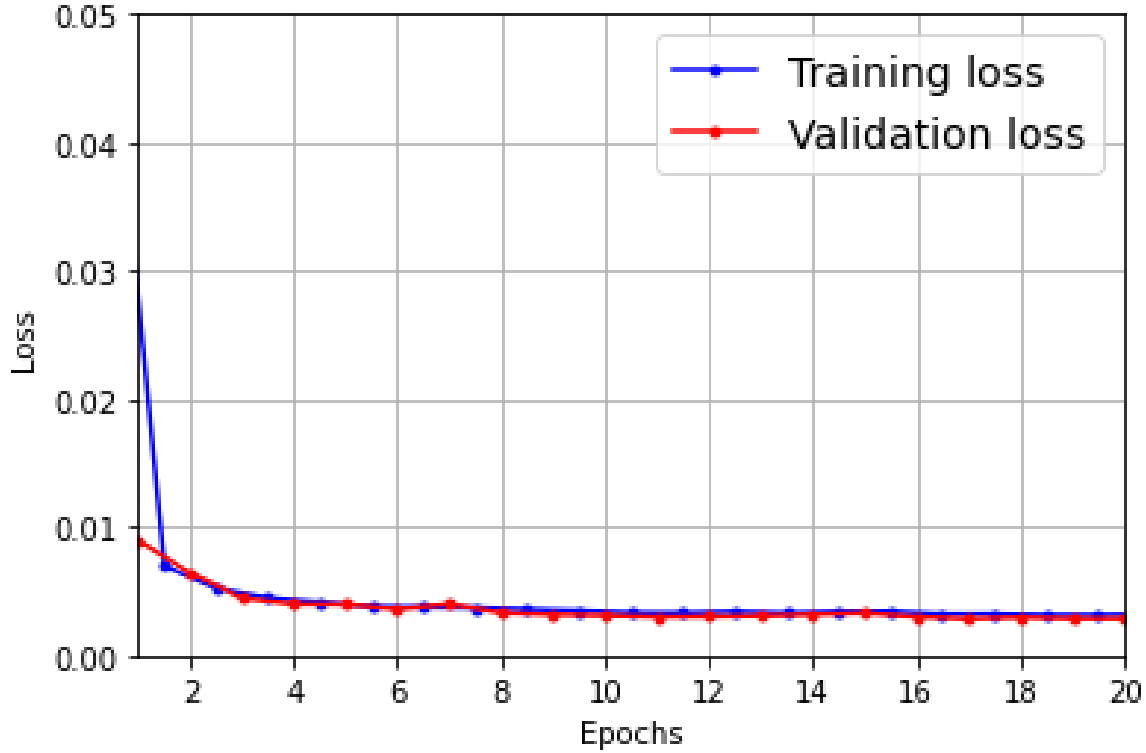
```
model.evaluate(X_valid, y_valid)
```

```
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
Epoch 93/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0110
Epoch 94/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
Epoch 95/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
Epoch 96/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
Epoch 97/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
Epoch 98/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
Epoch 99/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
Epoch 100/100
219/219 [==============================] - 1s 4ms/step - loss: 0.0114 - val_loss: 0.0109
63/63 [==============================] - 0s 1ms/step - loss: 0.0109

0.010865027084946632
```

Loss=0.0108

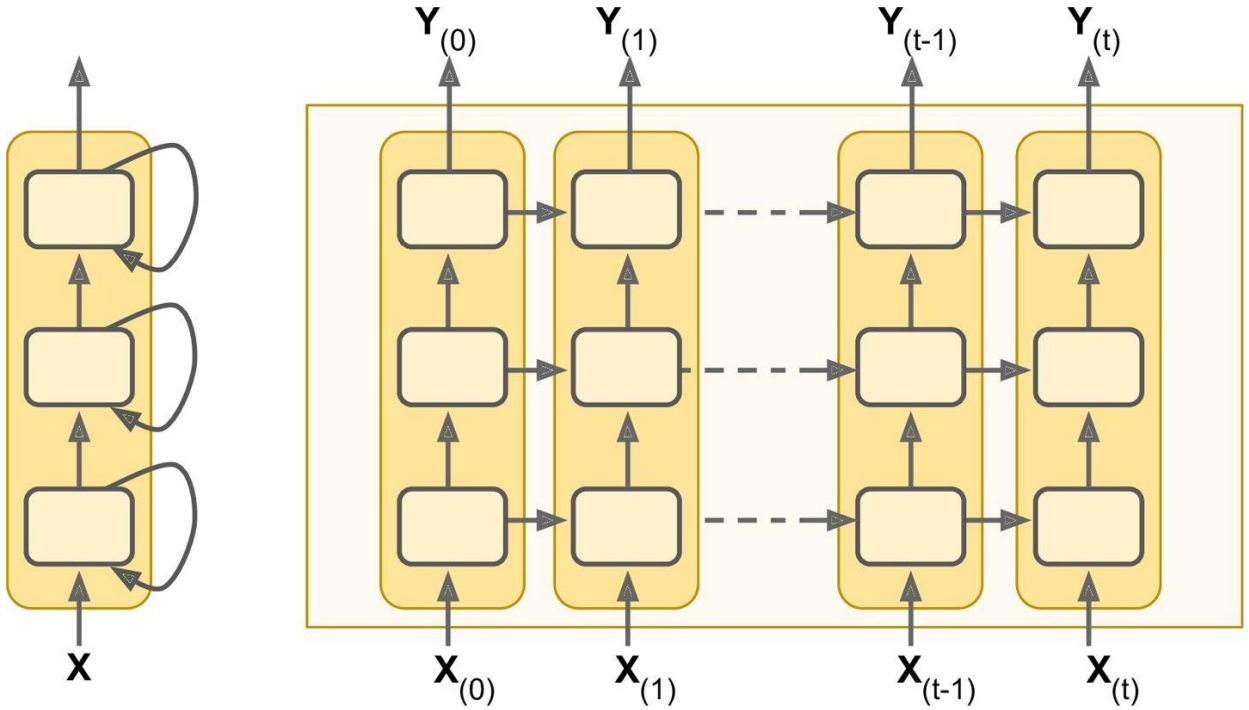
03. 간단한 RNN 모델


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x(t)
0
10
20
30
40
50
t


0.05
0.04
0.03
0.02
0.01
0.00
Loss
Training loss
Validation loss
2
4
6
8
10
12
14
16
18
20
Epochs

04. Deep RNN (A)

- 케라스에서 순환 층은 최종 출력만을 출력한다.
- 타임 스텝마다 출력을 반환하려면
  - ✓설정을 retrun_sequences=True

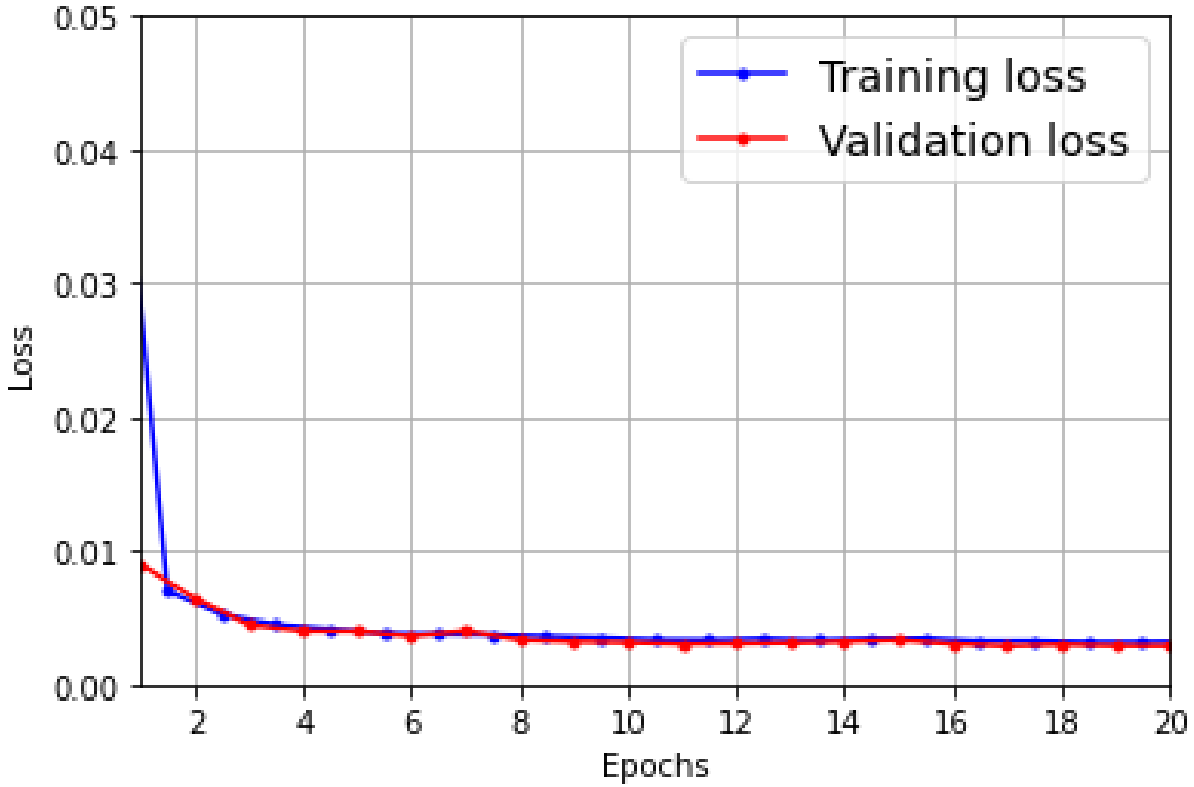


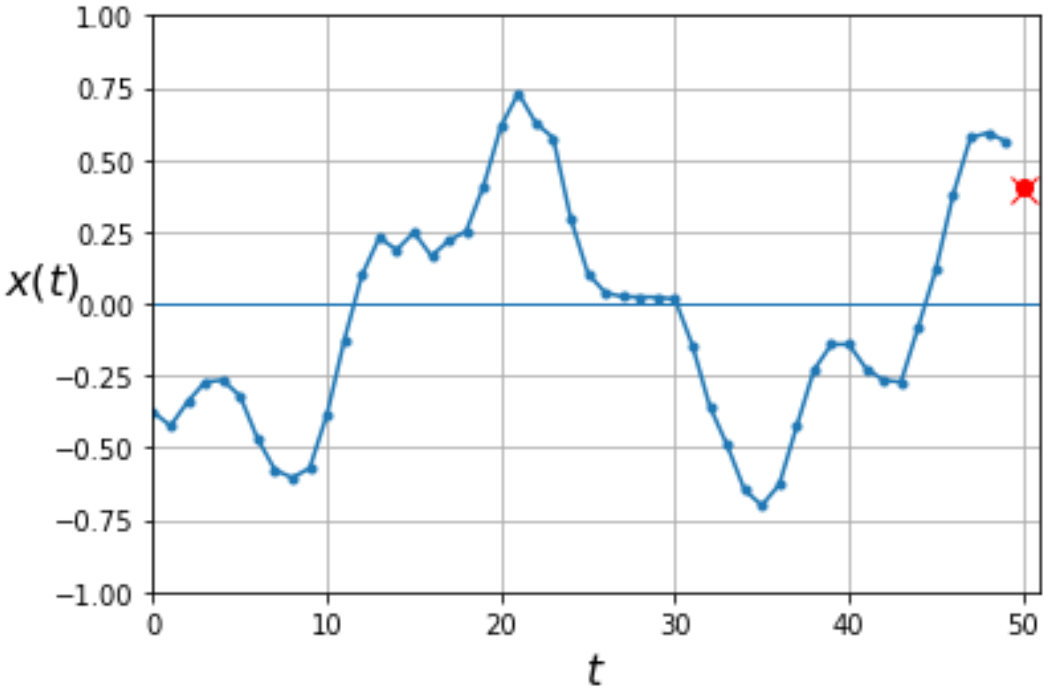
```
np.random.seed(42)
tf.random.set_seed(42)

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

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, y_train, epochs=100,
                    validation_data=(X_valid, y_valid))
```

```
Epoch 30/30
219/219 [==============================] - 3s 13ms/step - loss: 0.0030 - val_loss: 0.0028
```

```
model.evaluate(X_valid, y_valid)
```

```
63/63 [==============================] - 0s 4ms/step - loss: 0.0028

0.00280545512214303
```


Training loss
Validation loss
Loss
Epochs
0.05
0.04
0.03
0.02
0.01
0.00
2
4
6
8
10
12
14
16
18
20


x(t)
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50

05. Deep RNN (B)L Dense 층 사용

Loss=0.0026

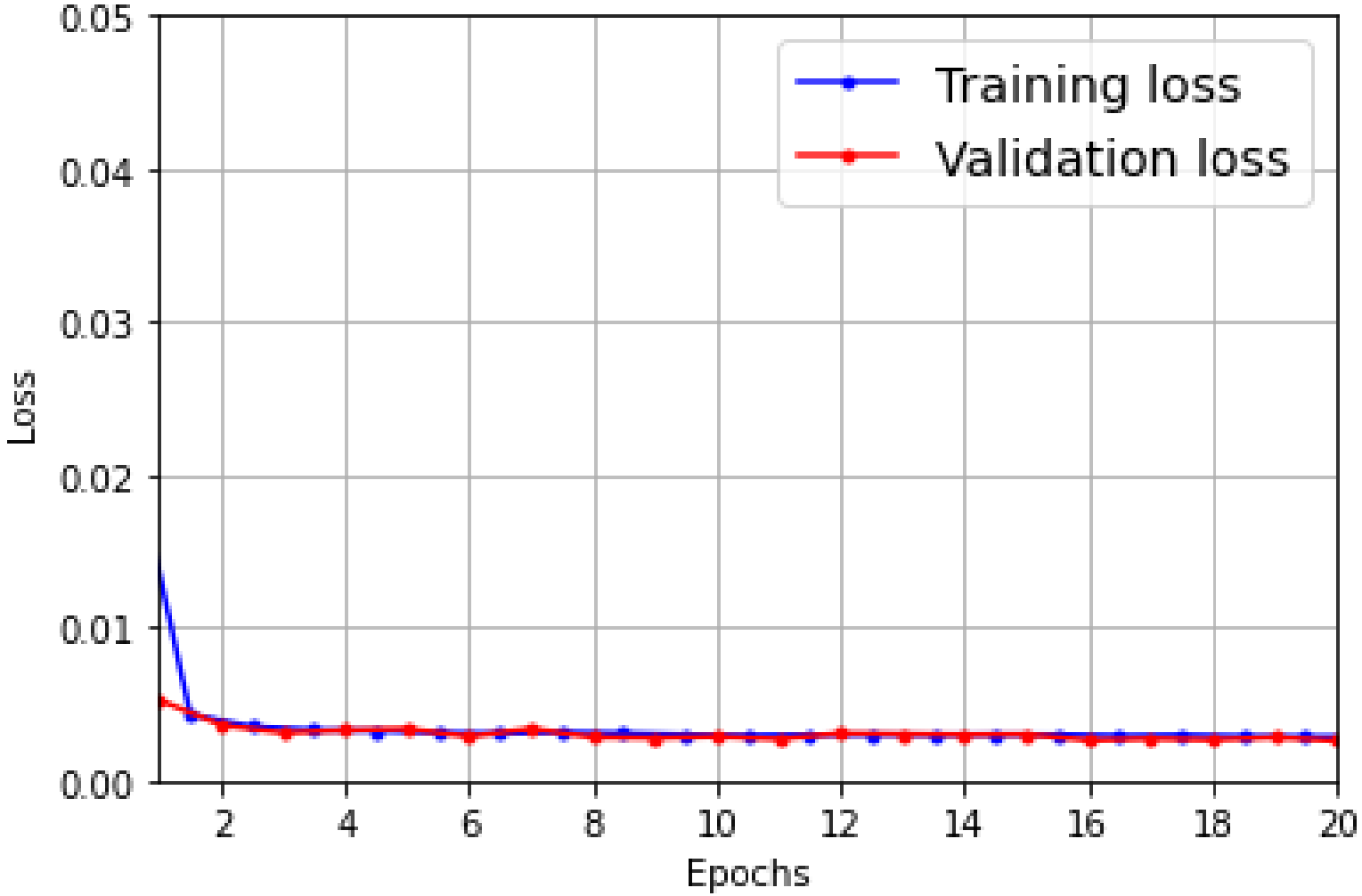
```
np.random.seed(42)
tf.random.set_seed(42)

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

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Epoch 30/30
219/219 [==============================] - 2s 9ms/step - loss: 0.0028 - val_loss: 0.0027
```

```
model.evaluate(X_valid, y_valid)
```

```
63/63 [==============================] - 0s 4ms/step - loss: 0.0027

0.0026875741314142942
```

05. Deep RNN (B)L Dense 층 사용



06 Many-to-One Prediction

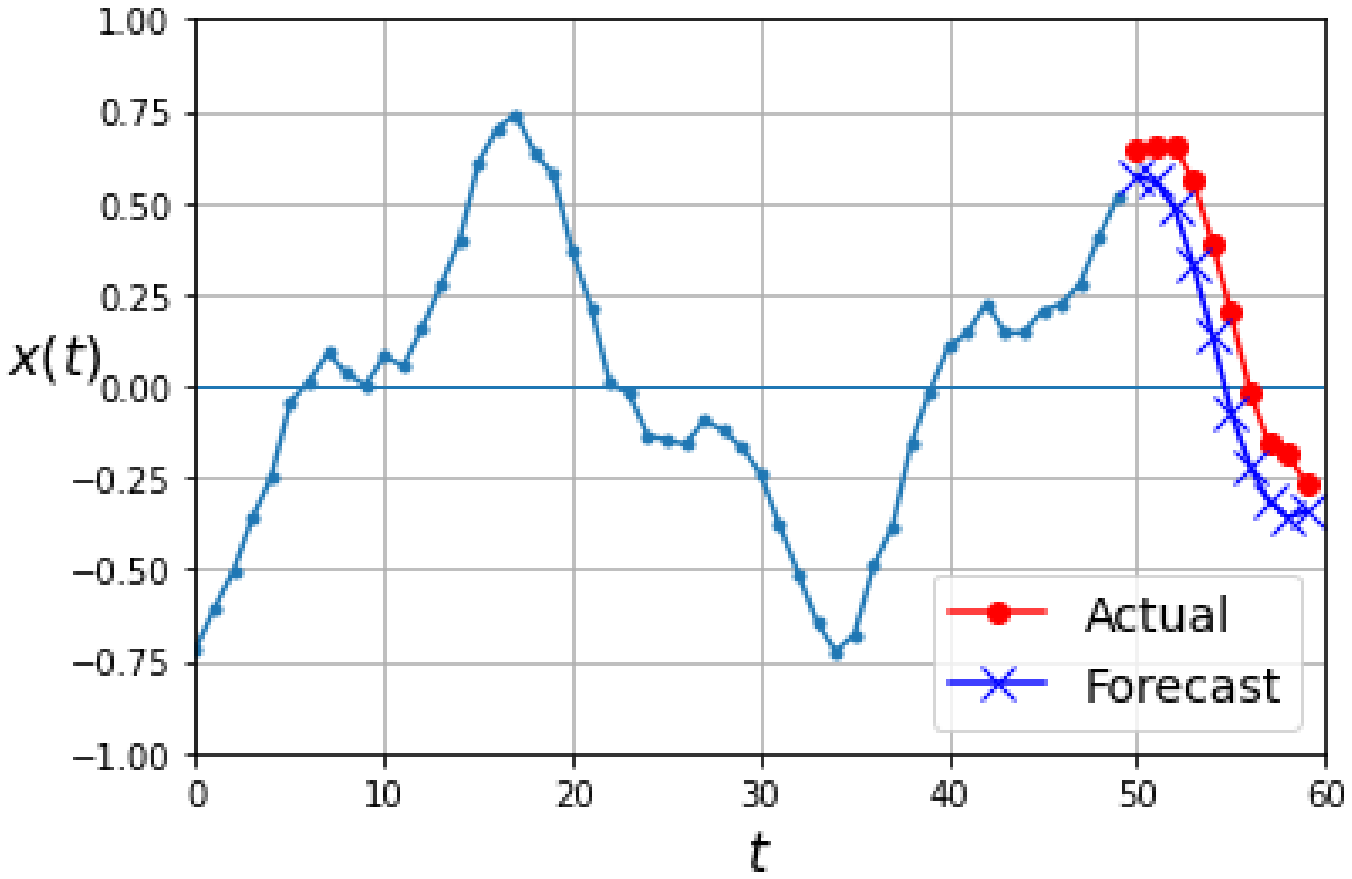
- 1개를 예측하는 것이 아니고 10개를 예측한다면
- 모델: 이미 훈련된 모델을 사용하여 다음 값을 예측하고, 그 다음 이 값을 입력으로 추가하여 다시 다음 값을 예측하는 방법이다.

```
np.random.seed(43) # not 42, as it would give the first series in th

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

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

```
Y_pred.shape
```

```
(1, 10, 1)
```

06 Many-to-One Prediction



06 Many-to-One Prediction

다음 10개 값을 예측하자. 타임 스텝 9개의 시퀀스를 만들자.

```
np.random.seed(42)

n_steps = 50
series = generate_time_series(10000, n_steps + 10)
X_train, Y_train = series[:7000, :n_steps], series[:7000, -10:, 0]
X_valid, Y_valid = series[7000:9000, :n_steps], series[7000:9000, -10:, 0]
X_test, Y_test = series[9000:, :n_steps], series[9000:, -10:, 0]
```

차근 차근 10개를 예측하지

```
X = X_valid
for step_ahead in range(10):
    y_pred_one = model.predict(X)[:, np.newaxis, :]
    X = np.concatenate([X, y_pred_one], axis=1)

Y_pred = X[:, n_steps:, 0]
```

```
#Y_pred.shape
```

```
np.mean(keras.metrics.mean_squared_error(Y_valid, Y_pred))
```

```
0.014123356
```

06 Many-to-One Prediction

이 성능과 기본 모델과 비교하자. naive model과 linear model

```
Y_naive_pred = Y_valid[:, -1:]
np.mean(keras.metrics.mean_squared_error(Y_valid, Y_naive_pred))
```

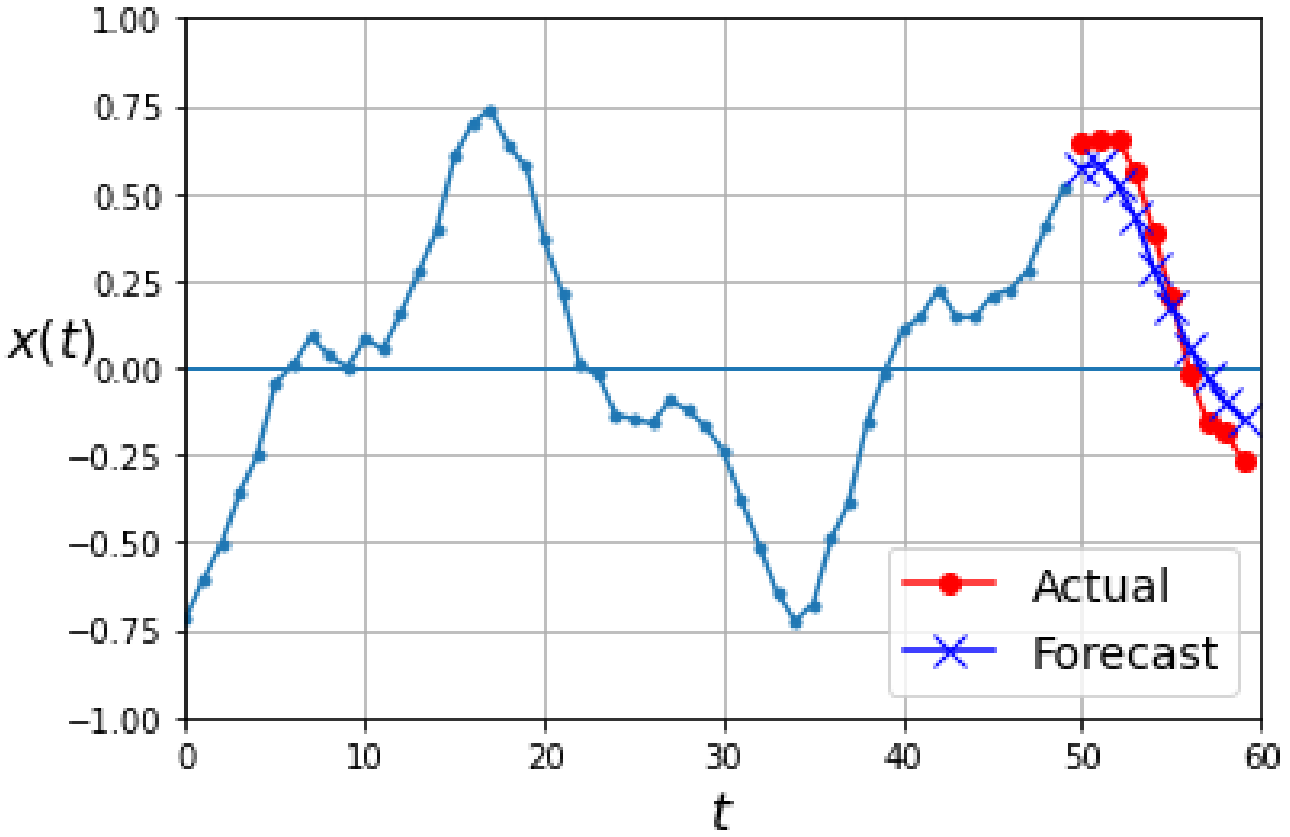
```
0.22278848
```

```
np.random.seed(42)
tf.random.set_seed(42)

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[50, 1]),
    keras.layers.Dense(10)
])

model.compile(loss="mse", optimizer="adam")
history = model.fit(X_train, Y_train, epochs=100,
                    validation_data=(X_valid, Y_valid))
```

```
model.evaluate(X_valid, y_valid)
```

```
63/63 [==============================] - 0s 1ms/step - loss: 0.2151

0.21506701409816742
```

06 Many-to-One Prediction



❖ 10개 앞을 RNN으로 예측하기

✓타입 스텝 0~49에서 타임 스텝 50~59를 예측하는 것이 아니다.

✓타입 스텝 0에서, 1~10을 예측하고, 타입스텝 2에서, 2~11을 예측하는 방법

```
np.random.seed(42)

n_steps = 50
series = generate_time_series(10000, n_steps + 10)
X_train = series[:7000, :n_steps]
X_valid = series[7000:9000, :n_steps]
X_test = series[9000:, :n_steps]
Y = np.empty((10000, n_steps, 10))
for step_ahead in range(1, 10 + 1):
    Y[..., step_ahead - 1] = series[..., step_ahead:step_ahead + n_steps, 0]
Y_train = Y[:7000]
Y_valid = Y[7000:9000]
Y_test = Y[9000:]
```

```
X_train.shape, Y_train.shape
```

```
((7000, 50, 1), (7000, 50, 10))
```

05-2 여러 타임 스텝 앞을 RNN으로 예측하기

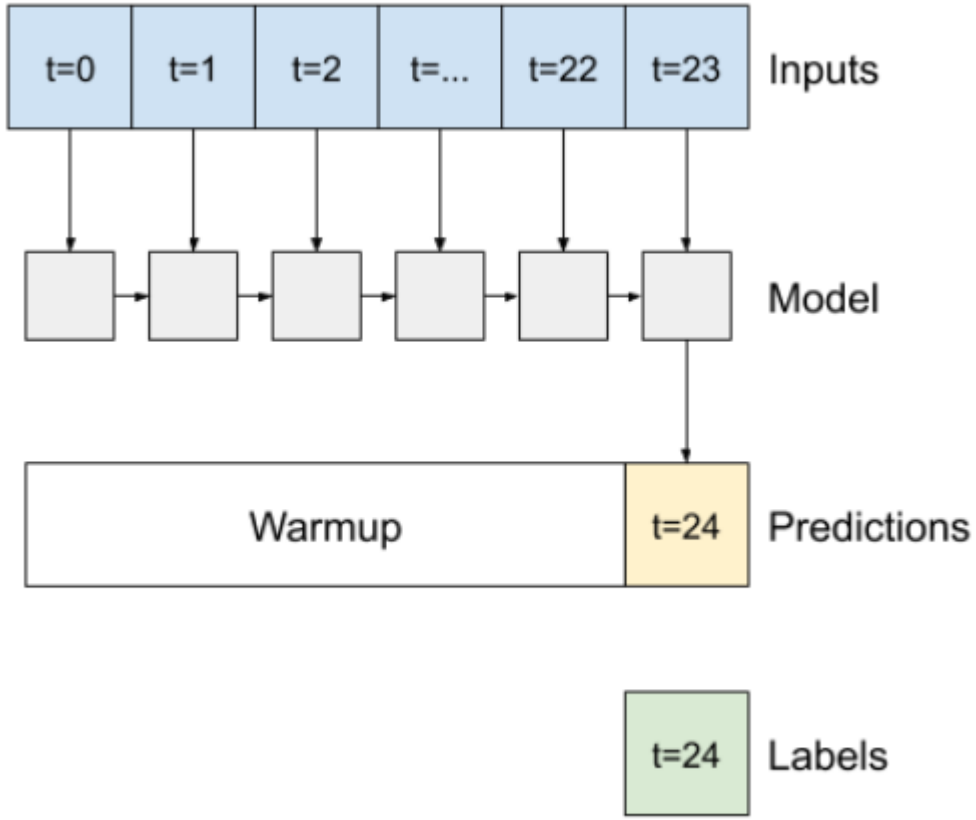
```
np.random.seed(42)
tf.random.set_seed(42)

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

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

model.compile(loss="mse", optimizer=keras.optimizers.Adam(lr=0.01),
              metrics=[last_time_step_mse])
history = model.fit(X_train, Y_train, epochs=100,
                    validation_data=(X_valid, Y_valid))
```

```
model.evaluate(X_valid, y_valid)
```

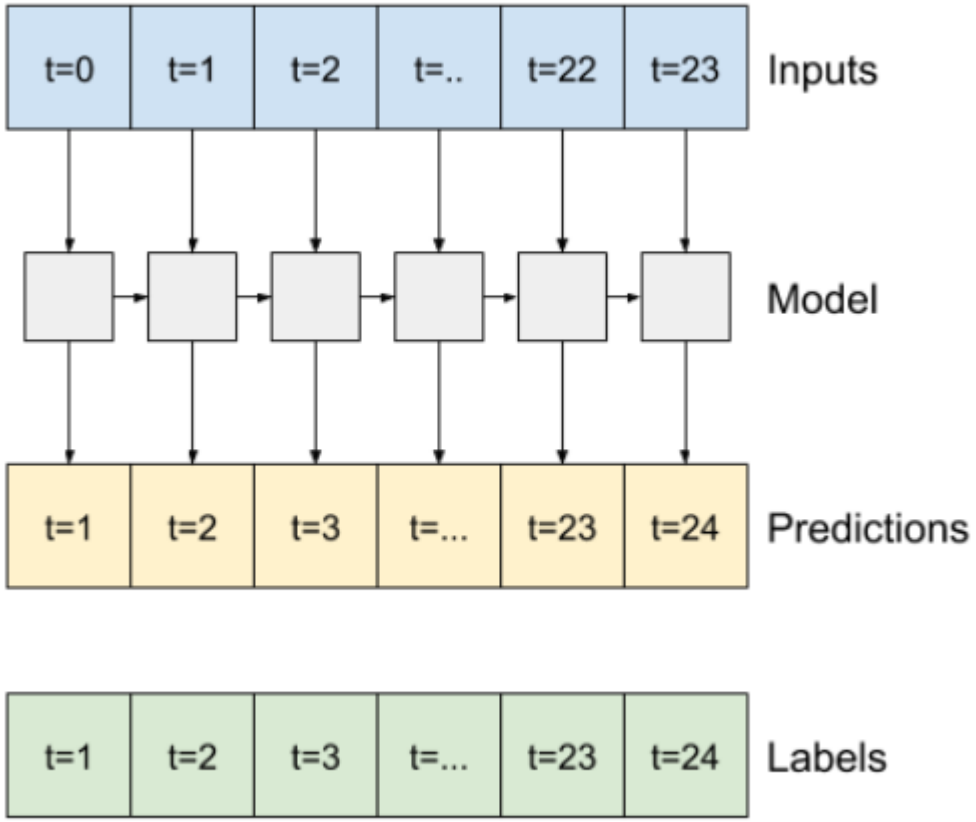
```
63/63 [==============================] - 0s 3ms/step - loss: 0.0079

0.007854317314922 81
```

차량검지기 데이터 분석

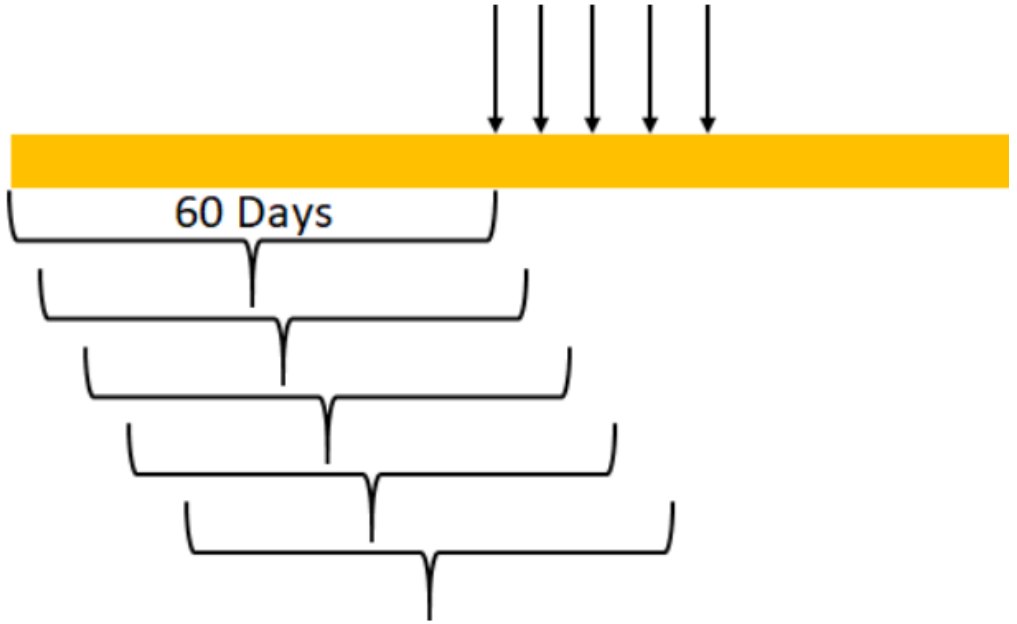
Recurrent Neural network



Stacking RNN layers



LSTM을 이용한 교통 흐름 예측



VDS 데이터를 이용한 교통 흐름 예측 모델 단계

- Data Preprocessing
- Building the RNN
- Making the prediction and visualization

VDS 데이터를 이용한 교통 흐름 예측 모델

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
from pandas import datetime
def parser(x):
    return datetime.strptime(x, '%Y-%m-%d %H:%M')
```

VDS 데이터를 이용한 교통 흐름 예측 모델

KISTI SINCE 1962

In [3]:
```
df = pd.read_csv('./daejeon_vds16.csv', date_parser=parser)
```

In [4]:
```
df.head()
```

Out [4] :

	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
0	2017-04-02 0:00	43	34	9	0	50.3	1.90
1	2017-04-02 0:05	45	32	13	0	58.9	1.84
2	2017-04-02 0:10	46	34	12	0	50.6	1.87
3	2017-04-02 0:15	45	36	9	0	50.9	1.72
4	2017-04-02 0:20	27	13	13	1	62.2	1.12

In [5]:
```
df.shape
```

Out [5] : (8064, 7)

정규화를 위하여 'Data' 특성을 제외하자

In [7]:
```
data_training  = df[df['Date'] <'2017-04-28'].copy()
data_test = df[df['Date'] >='2017-04-28'].copy()
```

Speed(속도) 데이터만을 선택해보자

In [8]:
```
data_training = data_training.drop(['Date','ToVol','SmVol','MeVol','LaVol','Occ.Rate'], axis = 1)
data_test = data_test.drop(['Date','ToVol','SmVol','MeVol','LaVol','Occ.Rate'], axis = 1)
```

In [9]:
```
data_training.head()
```

정규화

In [10]:
```
scaler = MinMaxScaler()
data_training = scaler.fit_transform(data_training)
data_test = scaler.fit_transform(data_test)
```

In [11]:
```
data_training.shape
```

Out[11]: (7488, 1)

In [12]:
```
data_test.shape
```

Out[12]: (576, 1)

RNN을 위하여 데이터를 재구성하자.

Loop_Back = 60으로 설정하자.

예측은 단일 예측 1개로 즉, 5분단위

create RNN with 60 timesteps, i.e. look 60 previous time steps

In [13]:
```
X_train = []
y_train = []
X_test = []
y_test = []
```

In [14]:
```
data_training.shape[0]
```

Out [14] : 7488

In [15]:
```
data_test.shape[0]
```

Out [15] : 576

train 데이터를 RNN 구조로 만들자

In [16]:
```
for i in range(60, data_training.shape[0]):
    X_train.append(data_training[i-60:i])
    y_train.append(data_training[i, 0])
```

In [17]:
```
X_train, y_train = np.array(X_train), np.array(y_train)
```

In [18]:
```
X_train.shape
```

Out [18] : (7428, 60, 1)

test 데이터에도 같은 구조로 해주자

In [19]:
```
for i in range(60, data_test.shape[0]):
    X_test.append(data_test[i-60:i])
    y_test.append(data_test[i, 0])
```

In [20]:
```
X_test, y_test = np.array(X_test), np.array(y_test)
```

Building LSTM

In [22]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
```

In [23]:
```
model = Sequential()
model.add(LSTM(units = 60, activation = 'relu', return_sequences = True,
               input_shape = (X_train.shape[1], 1)))
model.add(LSTM(units = 60, activation = 'sigmoid'))

model.add(Dense(units = 1))
```

Building LSTM

In [24]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 60)            14880

lstm_1 (LSTM)                (None, 60)                29040

dense (Dense)                (None, 1)                 61

=================================================================
Total params: 43,981
Trainable params: 43,981
Non-trainable params: 0
_________________________________________________________________
```

VDS 데이터를 이용한 교통 흐름 예측 모델

In [25]:
```
model.compile(optimizer='adam', loss = 'mean_squared_error',
              metrics =['accuracy'])
```

In [26]:
```
history= model.fit(X_train, y_train, epochs=5, batch_size=256)
```

```
Epoch 1/3
30/30 [==============================] - 26s 775ms/step - loss: 0.0914 - accuracy: 1.3463e-04
Epoch 2/3
30/30 [==============================] - 23s 754ms/step - loss: 0.0124 - accuracy: 1.3463e-04
Epoch 3/3
30/30 [==============================] - 23s 757ms/step - loss: 0.0092 - accuracy: 2.6925e-04
```

In [27]:
```
print(history.history)
```

```
{'loss': [0.09142845124006271, 0.012385959737002 85, 0.009211527183651924], 'accuracy': [0.00013462573406286538,
```

In [28]:
```
history.history.keys()
```

Out [28] :
```
dict_keys(['loss', 'accuracy'])
```

Prepare test dataset

In [29]:
```
X_test, y_test = np.array(X_test), np.array(y_test)
```

In [30]:
```
X_test.shape, y_test.shape
```

Out [30] : ((516, 60, 1), (516,))

In [41]:
```
model.evaluate(X_test, y_test, batch_size=128)
```

```
5/5 [==============================] - 0s 70ms/step - loss: 1387.8113 - accuracy: 0.0019
```

Out [41] : [1387.811279296875, 0.0019379844889044762]

In [31]:
```
y_pred = model.predict(X_test)
```

In [32]:
```
y_pred
```

Out [32] :
```
array([[0.5196103 ],
       [0.51958615],
```

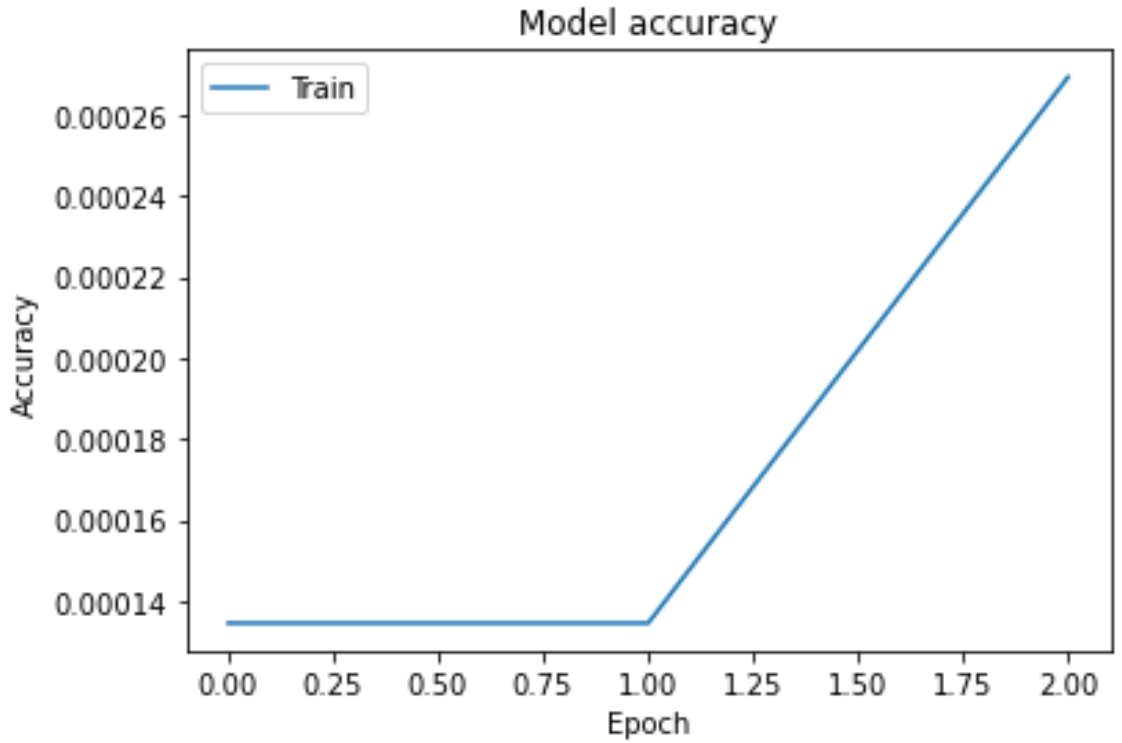
VDS 데이터를 이용한 교통 흐름 예측 모델

```
In [33]: scaler.scale_

Out [33] : array([0.01547988])

In [34]: #scale = 1/8.18605127e-04
         scale = 1/0.01547988

In [35]: y_pred = y_pred*scale
         y_test = y_test*scale

In [36]: y_pred
```

```
[33.666206],
[33.618874],
[33.71234 ],
```

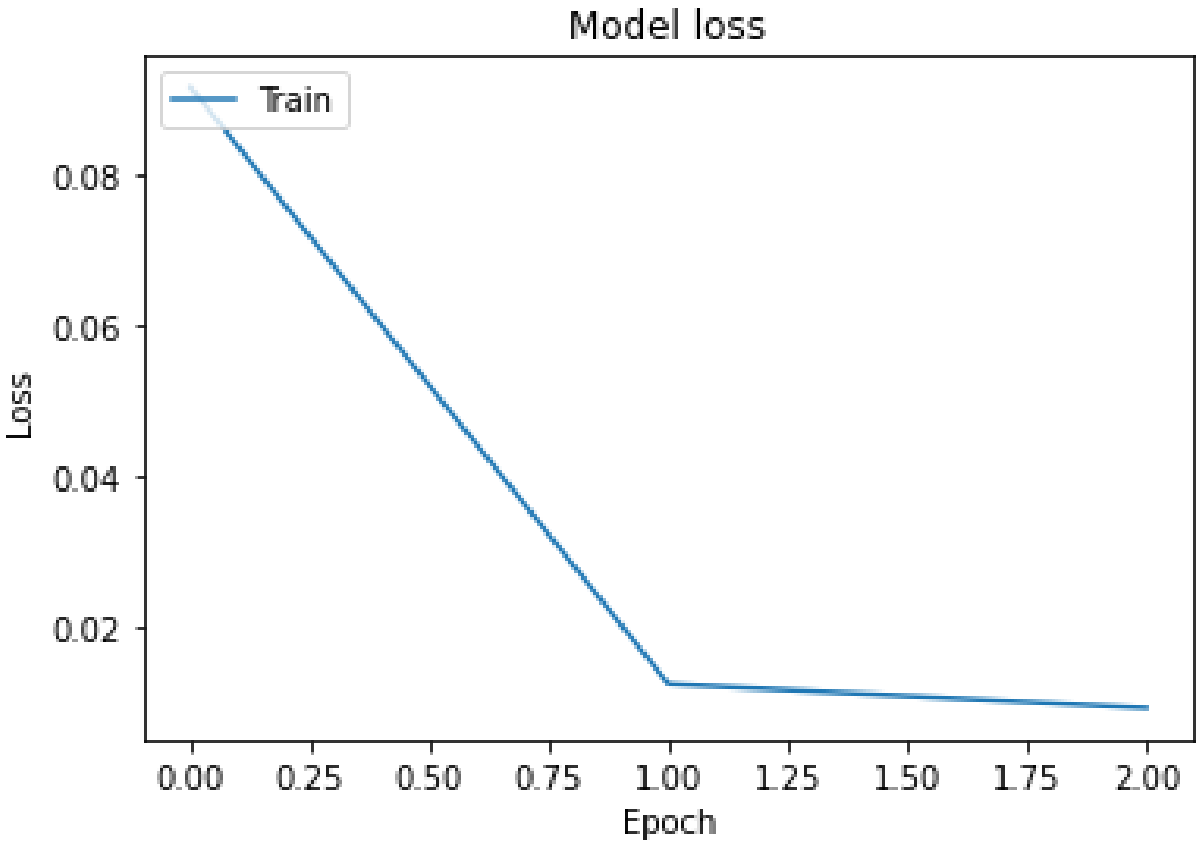
Visualization

[37]:
```
# 6 훈련 과정 시각화 (정확도)
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



VDS 데이터를 이용한 교통 흐름 예측 모델

In [38]:
```
# 7 훈련 과정 시각화 (손실)
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

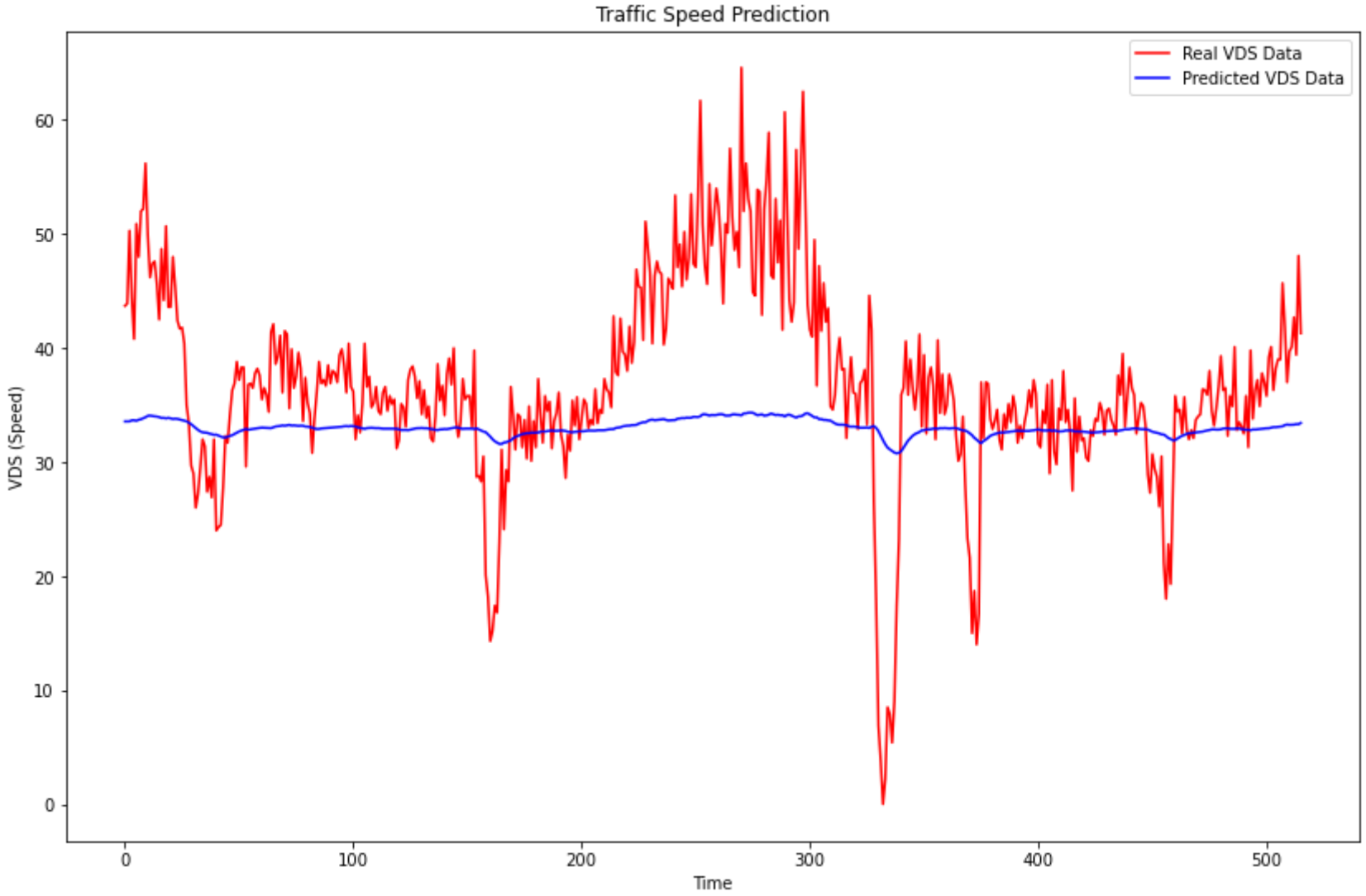


VDS 데이터를 이용한 교통 흐름 예측 모델

In [39]:

```
# Visualising the results
plt.figure(figsize=(14,5))
plt.plot(y_test, color = 'red', label = 'Real VDS Data')
plt.plot(y_pred, color = 'blue', label = 'Predicted VDS Data')
plt.title('Traffic Speed Prediction')
plt.xlabel('Time')
plt.ylabel('VDS (Speed)')
plt.legend()
plt.show()
```

VDS 데이터를 이용한 교통 흐름 예측 모델



2022 TRUST KISTI

Korea Institute of Science
and Technology Information