간단한 RNN을 구현해 보기

In [1]:

```
import tensorflow as tf
import keras
```

Out[1]:

```
'2.7.0'
```

In [2]:

```
print(tf.__version__)
print(keras.__version__)
```

```
2.7.0
2.7.0
```

In [3]:

```
import numpy as np
```

데이터 준비

In [7]:

```
for i in range(10):
    lst = list(range(i, i+5))
    print(lst)
```

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4, 5]
[2, 3, 4, 5, 6]
[3, 4, 5, 6, 7]
[4, 5, 6, 7, 8]
[5, 6, 7, 8, 9]
[6, 7, 8, 9, 10]
[7, 8, 9, 10, 11]
[8, 9, 10, 11, 12]
[9, 10, 11, 12, 13]
```

In [12]:

```python
X = []
Y = []
for i in range(10):
    lst = list(range(i, i+5))
    X.append( [ [c/10] for c in lst]  )  # 문제
    Y.append( (i+5)/10  )  #

X = np.array(X)
Y = np.array(Y)

print( X.shape, Y.shape ) # 10개의 샘플 (5,1), 다음 0.5
print( X[0], Y[0])
print( X[1], Y[1])
```

```
(10, 5, 1) (10,)
[[0. ]
 [0.1]
 [0.2]
 [0.3]
 [0.4]] 0.5
[[0.1]
 [0.2]
 [0.3]
 [0.4]
 [0.5]] 0.6
```

In [17]:

```
# 전체 데이터 확인
for i in range(len(X)):
    print(X[i], Y[i])
print( X.shape, Y.shape )
```

```
[[0. ]
 [0.1]
 [0.2]
 [0.3]
 [0.4]] 0.5
[[0.1]
 [0.2]
 [0.3]
 [0.4]
 [0.5]] 0.6
[[0.2]
 [0.3]
 [0.4]
 [0.5]
 [0.6]] 0.7
[[0.3]
 [0.4]
 [0.5]
 [0.6]
 [0.7]] 0.8
[[0.4]
 [0.5]
 [0.6]
 [0.7]
 [0.8]] 0.9
[[0.5]
 [0.6]
 [0.7]
 [0.8]
 [0.9]] 1.0
[[0.6]
 [0.7]
 [0.8]
 [0.9]
 [1. ]] 1.1
[[0.7]
 [0.8]
 [0.9]
 [1. ]
 [1.1]] 1.2
[[0.8]
 [0.9]
 [1. ]
 [1.1]
 [1.2]] 1.3
[[0.9]
 [1. ]
 [1.1]
 [1.2]
 [1.3]] 1.4
(10, 5, 1) (10,)
```

모델 구축

- SimpleRNN
  - return_sequences : 기본값(False)
    - False : 마지막 상태만 출력
    - True : 모든 지점의 은닉 상태 출력
  - return_state : 기본값(False)
    - True : return_sequences의 값과 상관없이 마지막 은닉 상태를 출력

In [18]:

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=10, return_sequences=False, input_shape=[5,1]),
    tf.keras.layers.Dense(1)
])
```

In [19]:

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 10)                120

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 131
Trainable params: 131
Non-trainable params: 0
_________________________________________________________________
```

파라미터 개수

- 파라미터 아웃 * (파라미터 아웃 + 차원 + 1(바이어스))
- 10 * (10 * 1 + 1) = 120

In [22]:

```
model.fit(X, Y, epochs=50, verbose=1)
```

```
Epoch 1/50
1/1 [==============================] - 0s 7ms/step - loss: 0.0011
Epoch 2/50
1/1 [==============================] - 0s 7ms/step - loss: 0.0011
Epoch 3/50
1/1 [==============================] - 0s 5ms/step - loss: 0.0010
Epoch 4/50
1/1 [==============================] - 0s 5ms/step - loss: 0.0010
Epoch 5/50
1/1 [==============================] - 0s 6ms/step - loss: 0.0010
Epoch 6/50
1/1 [==============================] - 0s 7ms/step - loss: 9.8520e-04
Epoch 7/50
1/1 [==============================] - 0s 10ms/step - loss: 9.6781e-04
Epoch 8/50
1/1 [==============================] - 0s 13ms/step - loss: 9.5067e-04
Epoch 9/50
1/1 [==============================] - 0s 10ms/step - loss: 9.3381e-04
Epoch 10/50
1/1 [==============================] - 0s 7ms/step - loss: 9.1720e-04
Epoch 11/50
1/1 [==============================] - 0s 8ms/step - loss: 9.0085e-04
Epoch 12/50
1/1 [==============================] - 0s 7ms/step - loss: 8.8476e-04
Epoch 13/50
1/1 [==============================] - 0s 10ms/step - loss: 8.6893e-04
Epoch 14/50
1/1 [==============================] - 0s 6ms/step - loss: 8.5334e-04
Epoch 15/50
1/1 [==============================] - 0s 7ms/step - loss: 8.3800e-04
Epoch 16/50
1/1 [==============================] - 0s 6ms/step - loss: 8.2291e-04
Epoch 17/50
1/1 [==============================] - 0s 6ms/step - loss: 8.0805e-04
Epoch 18/50
1/1 [==============================] - 0s 7ms/step - loss: 7.9343e-04
Epoch 19/50
1/1 [==============================] - 0s 6ms/step - loss: 7.7905e-04
Epoch 20/50
1/1 [==============================] - 0s 5ms/step - loss: 7.6491e-04
Epoch 21/50
1/1 [==============================] - 0s 5ms/step - loss: 7.5099e-04
Epoch 22/50
1/1 [==============================] - 0s 6ms/step - loss: 7.3730e-04
Epoch 23/50
1/1 [==============================] - 0s 7ms/step - loss: 7.2383e-04
Epoch 24/50
1/1 [==============================] - 0s 9ms/step - loss: 7.1060e-04
Epoch 25/50
1/1 [==============================] - 0s 13ms/step - loss: 6.9757e-04
Epoch 26/50
1/1 [==============================] - 0s 29ms/step - loss: 6.8476e-04
Epoch 27/50
1/1 [==============================] - 0s 9ms/step - loss: 6.7217e-04
Epoch 28/50
1/1 [==============================] - 0s 5ms/step - loss: 6.5979e-04
```

```
Epoch 29/50
1/1 [==============================] - 0s 6ms/step - loss: 6.4762e-04
Epoch 30/50
1/1 [==============================] - 0s 7ms/step - loss: 6.3565e-04
Epoch 31/50
1/1 [==============================] - 0s 7ms/step - loss: 6.2389e-04
Epoch 32/50
1/1 [==============================] - 0s 7ms/step - loss: 6.1232e-04
Epoch 33/50
1/1 [==============================] - 0s 7ms/step - loss: 6.0096e-04
Epoch 34/50
1/1 [==============================] - 0s 6ms/step - loss: 5.8979e-04
Epoch 35/50
1/1 [==============================] - 0s 10ms/step - loss: 5.7881e-04
Epoch 36/50
1/1 [==============================] - 0s 6ms/step - loss: 5.6802e-04
Epoch 37/50
1/1 [==============================] - 0s 8ms/step - loss: 5.5742e-04
Epoch 38/50
1/1 [==============================] - 0s 6ms/step - loss: 5.4701e-04
Epoch 39/50
1/1 [==============================] - 0s 9ms/step - loss: 5.3678e-04
Epoch 40/50
1/1 [==============================] - 0s 8ms/step - loss: 5.2673e-04
Epoch 41/50
1/1 [==============================] - 0s 11ms/step - loss: 5.1686e-04
Epoch 42/50
1/1 [==============================] - 0s 8ms/step - loss: 5.0716e-04
Epoch 43/50
1/1 [==============================] - 0s 13ms/step - loss: 4.9764e-04
Epoch 44/50
1/1 [==============================] - 0s 10ms/step - loss: 4.8829e-04
Epoch 45/50
1/1 [==============================] - 0s 6ms/step - loss: 4.7911e-04
Epoch 46/50
1/1 [==============================] - 0s 6ms/step - loss: 4.7009e-04
Epoch 47/50
1/1 [==============================] - 0s 6ms/step - loss: 4.6124e-04
Epoch 48/50
1/1 [==============================] - 0s 6ms/step - loss: 4.5255e-04
Epoch 49/50
1/1 [==============================] - 0s 6ms/step - loss: 4.4402e-04
Epoch 50/50
1/1 [==============================] - 0s 6ms/step - loss: 4.3565e-04
```

Out[22]:

```
<keras.callbacks.History at 0x7fbb02e5d750>
```

In [25]:

```
print(Y) # 실제값
pred = model.predict(X)
pred
```

```
[0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4]
```

Out[25]:

```
array([[0.4782341],
       [0.593213 ],
       [0.7048174],
       [0.8134223],
       [0.9188194],
       [1.0200301],
       [1.1155015],
       [1.2035673],
       [1.2828683],
       [1.3525735]], dtype=float32)
```