SimpleRNN Test Code

```
import warnings
warnings.filterwarnings('ignore')
```

Import Packages

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

I. SimpleRNN - without Normalization

1) Sample Data

- Inputs 데이터 생성(100, 5, 1)

```
X = [[[i + j] for i in range(5)] for j in range(100)]

X[:3], X[-3:]
```

```
([[[0], [1], [2], [3], [4]],
  [[1], [2], [3], [4], [5]],
  [[2], [3], [4], [5], [6]]],
 [[[97], [98], [99], [100], [101]],
  [[98], [99], [100], [101], [102]],
  [[99], [100], [101], [102], [103]]])
```

- Outputs 데이터 생성(100, 1)

```
y = [(i + 5) for i in range(100)]

y[:3], y[-3:]
```

```
([5, 6, 7], [102, 103, 104])
```

2) numpy_Array Casting

```
X = np.array(X, dtype = float)
y = np.array(y, dtype = float)

X.shape, y.shape
```

```
((100, 5, 1), (100,))
```

3) Train vs. Test Split

- 80:20

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    random_state = 2045)

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((80, 5, 1), (80,), (20, 5, 1), (20,))
```

4) Keras SimpleRNN Modeling

(1) Model Define & Summary

- Unit(output_dim) : 3
- input_shape(input_lenght, input_dim) : (5, 1)
- return_sequences = False : 최종 Unit만 출력
- layers.Dense(1) : y_hat

```
from keras import models, layers

model_1 = models.Sequential(name = 'SimpleRNN_1')
model_1.add(layers.SimpleRNN(3,
                             input_shape = (5, 1),
                             return_sequences = False))
model_1.add(layers.Dense(1))

model_1.summary()
```

```
Model: "SimpleRNN_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 3)                 15
_________________________________________________________________
dense (Dense)                (None, 1)                 4
```

```
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
```

(2) Model Compile

```
model_1.compile(loss = 'mse',
                optimizer = 'adam',
                metrics = ['accuracy'])
```

(3) Model Fit

```
Hist_1 = model_1.fit(X_train, y_train,
                     epochs = 100,
                     batch_size = 8,
                     validation_data = (X_test, y_test))
```

```
Epoch 1/100
10/10 [==============================] - 1s 55ms/step - loss: 3478.0078 - accuracy: 0.0000
Epoch 2/100
10/10 [==============================] - 0s 5ms/step - loss: 4047.5542 - accuracy: 0.0000e
Epoch 3/100
10/10 [==============================] - 0s 5ms/step - loss: 3452.3417 - accuracy: 0.0000e
Epoch 4/100
10/10 [==============================] - 0s 5ms/step - loss: 3893.3910 - accuracy: 0.0000e
Epoch 5/100
10/10 [==============================] - 0s 5ms/step - loss: 3838.1765 - accuracy: 0.0000e
Epoch 6/100
10/10 [==============================] - 0s 6ms/step - loss: 3646.7904 - accuracy: 0.0000e
Epoch 7/100
10/10 [==============================] - 0s 5ms/step - loss: 3862.8047 - accuracy: 0.0000e
Epoch 8/100
10/10 [==============================] - 0s 5ms/step - loss: 3796.7033 - accuracy: 0.0000e
Epoch 9/100
10/10 [==============================] - 0s 5ms/step - loss: 4128.7536 - accuracy: 0.0000e
Epoch 10/100
10/10 [==============================] - 0s 5ms/step - loss: 3829.4810 - accuracy: 0.0000e
Epoch 11/100
10/10 [==============================] - 0s 5ms/step - loss: 3880.8670 - accuracy: 0.0000e
Epoch 12/100
10/10 [==============================] - 0s 5ms/step - loss: 3944.1355 - accuracy: 0.0000e
Epoch 13/100
10/10 [==============================] - 0s 5ms/step - loss: 3782.6383 - accuracy: 0.0000e
Epoch 14/100
10/10 [==============================] - 0s 6ms/step - loss: 3502.3135 - accuracy: 0.0000e
Epoch 15/100
10/10 [==============================] - 0s 6ms/step - loss: 4250.8127 - accuracy: 0.0000e
Epoch 16/100
10/10 [==============================] - 0s 5ms/step - loss: 3849.8813 - accuracy: 0.0000e
Epoch 17/100
10/10 [==============================] - 0s 5ms/step - loss: 3704.2609 - accuracy: 0.0000e
Epoch 18/100
```

```
10/10 [==============================] - 0s 5ms/step - loss: 4190.0792 - accuracy: 0.0000e
Epoch 19/100
10/10 [==============================] - 0s 6ms/step - loss: 3886.7507 - accuracy: 0.0000e
Epoch 20/100
10/10 [==============================] - 0s 6ms/step - loss: 3289.0334 - accuracy: 0.0000e
Epoch 21/100
10/10 [==============================] - 0s 5ms/step - loss: 3412.4140 - accuracy: 0.0000e
Epoch 22/100
10/10 [==============================] - 0s 5ms/step - loss: 4151.6605 - accuracy: 0.0000e
Epoch 23/100
10/10 [==============================] - 0s 5ms/step - loss: 4175.3132 - accuracy: 0.0000e
Epoch 24/100
10/10 [==============================] - 0s 5ms/step - loss: 3512.4541 - accuracy: 0.0000e
Epoch 25/100
10/10 [==============================] - 0s 5ms/step - loss: 4177.6090 - accuracy: 0.0000e
Epoch 26/100
10/10 [==============================] - 0s 6ms/step - loss: 3869.0067 - accuracy: 0.0000e
Epoch 27/100
10/10 [==============================] - 0s 5ms/step - loss: 3986.0249 - accuracy: 0.0000e
Epoch 28/100
10/10 [==============================] - 0s 5ms/step - loss: 3920.7421 - accuracy: 0.0000e
Epoch 29/100
```

(4) Model Predict

```
y_hat = model_1.predict(X_test)
```

(5) 학습 결과 시각화

- Loss 감소

```
plt.plot(Hist_1.history['loss'])
plt.show()
```

- 학습 되지 않음
  - 녹색 -> 정답(y_test)
  - 적색 -> 예측(y_hat)

```
plt.scatter(range(20), y_hat, c = 'r')
plt.scatter(range(20), y_test, c = 'g')
plt.show()
```

II. SimpleRNN - with Normalization

1) Sample Data - with Normalization

```
X = [[[(i + j)] for i in range(5)] for j in range(100)]
y = [(i + 5) for i in range(100)]

X = (X - np.min(X)) / (np.max(X) - np.min(X))
y = (y - np.min(y)) / (np.max(y) - np.min(y))
```

2) Casting

```
X = np.array(X, dtype = float)
y = np.array(y, dtype = float)

X.shape, y.shape
```

```
((100, 5, 1), (100,))
```

3) Train vs. Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    random_state = 2045)

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((80, 5, 1), (80,), (20, 5, 1), (20,))
```

4) Keras SimpleRNN Modeling

(1) Model Define & Summary

- None : input_length 자동 맞춤

```
model_2 = models.Sequential(name = 'SimpleRNN_2')
model_2.add(layers.SimpleRNN(3,
                             input_shape = (None, 1),
                             return_sequences = False))
model_2.add(layers.Dense(1))

model_2.summary()
```

```
Model: "SimpleRNN_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 3)                 15
```

```
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 4         
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
_________________________________________________________________
```

(2) Model Compile

```
model_2.compile(loss = 'mse',
                optimizer = 'adam',
                metrics = ['accuracy'])
```

(3) Model Fit

```
Hist_2 = model_2.fit(X_train, y_train,
                     epochs = 100,
                     batch_size = 8,
                     validation_data = (X_test, y_test))
```

```
Epoch 1/100
10/10 [==============================] - 1s 28ms/step - loss: 0.2848 - accuracy: 0.0231 -
Epoch 2/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2517 - accuracy: 0.0136 - v
Epoch 3/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2637 - accuracy: 0.0231 - v
Epoch 4/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2464 - accuracy: 0.0231 - v
Epoch 5/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1905 - accuracy: 0.0344 - v
Epoch 6/100
10/10 [==============================] - 0s 6ms/step - loss: 0.1951 - accuracy: 0.0050 - v
Epoch 7/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1732 - accuracy: 0.0107 - v
Epoch 8/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1453 - accuracy: 0.0344 - v
Epoch 9/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1622 - accuracy: 0.0136 - v
Epoch 10/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1169 - accuracy: 0.0035 - v
Epoch 11/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1210 - accuracy: 0.0231 - v
Epoch 12/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0683 - accuracy: 0.0066 - v
Epoch 13/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0984 - accuracy: 0.0066 - v
Epoch 14/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0709 - accuracy: 0.0023 - v
Epoch 15/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0737 - accuracy: 0.0174 - v
Epoch 16/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0598 - accuracy: 0.0050 - v
Epoch 17/100
```

```
10/10 [==============================] - 0s 5ms/step - loss: 0.0513 - accuracy: 0.0136 - v
Epoch 18/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0503 - accuracy: 0.0173 - v
Epoch 19/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0566 - accuracy: 0.0380 - v
Epoch 20/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0453 - accuracy: 0.0280 - v
Epoch 21/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0410 - accuracy: 0.0115 - v
Epoch 22/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0306 - accuracy: 0.0380 - v
Epoch 23/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0313 - accuracy: 0.0575 - v
Epoch 24/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0227 - accuracy: 0.0196 - v
Epoch 25/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0197 - accuracy: 0.0243 - v
Epoch 26/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0195 - accuracy: 0.0157 - v
Epoch 27/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0184 - accuracy: 0.0338 - v
Epoch 28/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0168 - accuracy: 0.0151 - v
Epoch 29/100
```

(4) Model Predict

```
y_hat = model_2.predict(X_test)
```

(5) 학습 결과 시각화

- Loss 감소

```
plt.plot(Hist_2.history['loss'])
plt.show()
```

- 학습 진행
  - 녹색 -> 정답(y_test)
  - 적색 -> 예측(y_hat)

```
plt.scatter(range(20), y_hat, c = 'r')
plt.scatter(range(20), y_test, c = 'g')
plt.show()
```

III. Stacked_SimpleRNN

1) Model Define & Summary

- return_sequences = True

```
model_4 = models.Sequential(name = 'Stackd_RNN')
model_4.add(layers.SimpleRNN(3,
                             input_shape = (None, 1),
                             return_sequences = True))
model_4.add(layers.SimpleRNN(3,
                             input_shape = (None, 1),
                             return_sequences = False))
model_4.add(layers.Dense(1))

model_4.summary()
```

```
Model: "Stackd_RNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_2 (SimpleRNN)     (None, None, 3)           15
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 3)                 21
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 4
=================================================================
Total params: 40
Trainable params: 40
Non-trainable params: 0
_________________________________________________________________
```

2) Model Compile

```
model_4.compile(loss = 'mse',
                optimizer = 'adam',
                metrics = ['accuracy'])
```

3) Model Fit

```
Hist_4 = model_4.fit(X_train, y_train,
                     epochs = 100,
                     batch_size = 8,
                     validation_data = (X_test, y_test))
```

```
Epoch 1/100
10/10 [==============================] - 1s 34ms/step - loss: 1.2466 - accuracy: 0.0066 -
Epoch 2/100
10/10 [==============================] - 0s 6ms/step - loss: 0.7573 - accuracy: 0.0066 - v
Epoch 3/100
```

```
10/10 [==============================] - 0s 5ms/step - loss: 0.3146 - accuracy: 0.0000e+00
Epoch 4/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1746 - accuracy: 0.0000e+00
Epoch 5/100
10/10 [==============================] - 0s 6ms/step - loss: 0.1553 - accuracy: 0.0050 - v
Epoch 6/100
10/10 [==============================] - 0s 6ms/step - loss: 0.1328 - accuracy: 0.0231 - v
Epoch 7/100
10/10 [==============================] - 0s 6ms/step - loss: 0.1219 - accuracy: 0.0344 - v
Epoch 8/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0901 - accuracy: 0.0221 - v
Epoch 9/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0728 - accuracy: 0.0115 - v
Epoch 10/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0745 - accuracy: 0.0367 - v
Epoch 11/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0515 - accuracy: 0.0281 - v
Epoch 12/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0373 - accuracy: 0.0101 - v
Epoch 13/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0288 - accuracy: 0.0429 - v
Epoch 14/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0252 - accuracy: 0.0280 - v
Epoch 15/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0208 - accuracy: 0.0281 - v
Epoch 16/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0140 - accuracy: 0.0107 - v
Epoch 17/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0110 - accuracy: 0.0209 - v
Epoch 18/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0083 - accuracy: 0.0085 - v
Epoch 19/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0082 - accuracy: 0.0266 - v
Epoch 20/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0077 - accuracy: 0.0575 - v
Epoch 21/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0069 - accuracy: 0.0240 - v
Epoch 22/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0068 - accuracy: 0.0045 - v
Epoch 23/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0064 - accuracy: 0.0143 - v
Epoch 24/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0059 - accuracy: 0.0209 - v
Epoch 25/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0056 - accuracy: 0.0366 - v
Epoch 26/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0063 - accuracy: 0.0280 - v
Epoch 27/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0053 - accuracy: 0.0253 - v
Epoch 28/100
10/10 [==============================] - 0s 6ms/step - loss: 0.0049 - accuracy: 0.0169 - v
Epoch 29/100
```

4) Model Predict

```
y_hat = model_4.predict(X_test)
```

5) 학습 결과 시각화

```
plt.plot(Hist_4.history['loss'])
plt.show()
```

```
plt.scatter(range(20), y_hat, c = 'r')
plt.scatter(range(20), y_test, c = 'g')
plt.show()
```

IV. 'return_sequences' Output_Options

- 'input_length'에 대한 Sequance 전체를 출력할지 설정
  - 'False' vs. 'True'

1) 실습데이터 생성

```
X = [[[i + j] for i in range(5)] for j in range(100)]
y = [i + 5 for i in range(100)]
```

```
X = np.array(X, dtype = float)
y = np.array(y, dtype = float)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    random_state = 2045)

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((80, 5, 1), (80,), (20, 5, 1), (20,))
```

2) 테스트용 Input Data

```
X_test[0].reshape(1, 5, 1)
```

```
array([[[29.],
        [30.],
        [31.],
        [32.],
        [33.]]])
```

3) False_Option

- 마지막 Output만 출력
  - Unit -> 1

```
Model_False = models.Sequential()
Model_False.add(layers.SimpleRNN(1,
                                 input_shape = (5, 1),
                                 return_sequences = False))

Model_False.compile(loss = 'mse',
                    optimizer = 'adam',
                    metrics = ['accuracy'])
```

```
Model_False.predict(X_test[0].reshape(1, 5, 1))
```

```
array([[-1.]], dtype=float32)
```

- 마지막 Output만 출력
  - Unit -> 3

```
Model_False = models.Sequential()
Model_False.add(layers.SimpleRNN(3,
                                 input_shape = (5, 1),
                                 return_sequences = False))

Model_False.compile(loss = 'mse',
                    optimizer = 'adam',
                    metrics = ['accuracy'])
```

```
Model_False.predict(X_test[0].reshape(1, 5, 1))
```

```
array([[-1., -1., -1.]], dtype=float32)
```

4) True_Option

- 매 순환마다 Output 출력
  - Unit -> 1
  - input_length -> 5

```
Model_True = models.Sequential()
Model_True.add(layers.SimpleRNN(1,
                                input_shape = (5, 1),
                                return_sequences = True))

Model_True.compile(loss = 'mse',
                   optimizer = 'adam',
                   metrics = ['accuracy'])
```

```
                  metrics = ['accuracy'])
```

```
Model_True.predict(X_test[0].reshape(1, 5, 1))
```

```
array([[[1.],
        [1.],
        [1.],
        [1.],
        [1.]]], dtype=float32)
```

- 매 순환마다 Output만 출력
  - Unit -> 3
  - input_length -> 5

```
Model_True = models.Sequential()
Model_True.add(layers.SimpleRNN(3,
                                input_shape = (5, 1),
                                return_sequences = True))

Model_True.compile(loss = 'mse',
                   optimizer = 'adam',
                   metrics = ['accuracy'])
```

```
Model_True.predict(X_test[0].reshape(1, 5, 1))
```

```
array([[[ 1.,  1., -1.],
        [ 1.,  1., -1.],
        [ 1.,  1., -1.],
        [ 1.,  1., -1.],
        [ 1.,  1., -1.]]], dtype=float32)
```

#

#

#

The End

#

#

#