# 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 tensorflow.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 22ms/step - loss: 3450.6641 - accuracy: 0.0000e+00 - val_loss: 3397.5671 - val_accuracy: 0.00(
Epoch 2/100
10/10 [==============================] - 0s 3ms/step - loss: 3446.7063 - accuracy: 0.0000e+00 - val_loss: 3393.4734 - val_accuracy: 0.000(
Epoch 3/100
10/10 [==============================] - 0s 3ms/step - loss: 3442.6538 - accuracy: 0.0000e+00 - val_loss: 3389.4155 - val_accuracy: 0.000(
Epoch 4/100
10/10 [==============================] - 0s 3ms/step - loss: 3438.7065 - accuracy: 0.0000e+00 - val_loss: 3385.3418 - val_accuracy: 0.000(
Epoch 5/100
10/10 [==============================] - 0s 3ms/step - loss: 3434.7505 - accuracy: 0.0000e+00 - val_loss: 3381.2781 - val_accuracy: 0.000(
Epoch 6/100
10/10 [==============================] - 0s 3ms/step - loss: 3430.7129 - accuracy: 0.0000e+00 - val_loss: 3377.3047 - val_accuracy: 0.000(
Epoch 7/100
10/10 [==============================] - 0s 3ms/step - loss: 3426.7207 - accuracy: 0.0000e+00 - val_loss: 3373.3413 - val_accuracy: 0.000(
Epoch 8/100
10/10 [==============================] - 0s 3ms/step - loss: 3422.9067 - accuracy: 0.0000e+00 - val_loss: 3369.2488 - val_accuracy: 0.000(
Epoch 9/100
10/10 [==============================] - 0s 3ms/step - loss: 3418.8452 - accuracy: 0.0000e+00 - val_loss: 3365.2571 - val_accuracy: 0.000(
Epoch 10/100
10/10 [==============================] - 0s 3ms/step - loss: 3414.9199 - accuracy: 0.0000e+00 - val_loss: 3361.2422 - val_accuracy: 0.000(
Epoch 11/100
10/10 [==============================] - 0s 3ms/step - loss: 3410.9761 - accuracy: 0.0000e+00 - val_loss: 3357.2239 - val_accuracy: 0.000(
Epoch 12/100
```

```
10/10 [==============================] - 0s 4ms/step - loss: 3407.0305 - accuracy: 0.0000e+00 - val_loss: 3353.2141 - val_accuracy: 0.000(
Epoch 13/100
10/10 [==============================] - 0s 3ms/step - loss: 3403.1372 - accuracy: 0.0000e+00 - val_loss: 3349.1746 - val_accuracy: 0.000(
Epoch 14/100
10/10 [==============================] - 0s 4ms/step - loss: 3399.1094 - accuracy: 0.0000e+00 - val_loss: 3345.2231 - val_accuracy: 0.000(
Epoch 15/100
10/10 [==============================] - 0s 3ms/step - loss: 3395.2710 - accuracy: 0.0000e+00 - val_loss: 3341.1863 - val_accuracy: 0.000(
Epoch 16/100
10/10 [==============================] - 0s 4ms/step - loss: 3391.3040 - accuracy: 0.0000e+00 - val_loss: 3337.1929 - val_accuracy: 0.000(
Epoch 17/100
10/10 [==============================] - 0s 4ms/step - loss: 3387.3918 - accuracy: 0.0000e+00 - val_loss: 3333.1909 - val_accuracy: 0.000(
Epoch 18/100
10/10 [==============================] - 0s 4ms/step - loss: 3383.3706 - accuracy: 0.0000e+00 - val_loss: 3329.2898 - val_accuracy: 0.000(
Epoch 19/100
10/10 [==============================] - 0s 3ms/step - loss: 3379.5659 - accuracy: 0.0000e+00 - val_loss: 3325.2798 - val_accuracy: 0.000(
Epoch 20/100
10/10 [==============================] - 0s 3ms/step - loss: 3375.6250 - accuracy: 0.0000e+00 - val_loss: 3321.3054 - val_accuracy: 0.000(
Epoch 21/100
10/10 [==============================] - 0s 4ms/step - loss: 3371.7515 - accuracy: 0.0000e+00 - val_loss: 3317.3040 - val_accuracy: 0.000(
Epoch 22/100
10/10 [==============================] - 0s 3ms/step - loss: 3367.8621 - accuracy: 0.0000e+00 - val_loss: 3313.2993 - val_accuracy: 0.000(
Epoch 23/100
10/10 [==============================] - 0s 3ms/step - loss: 3363.8813 - accuracy: 0.0000e+00 - val_loss: 3309.3796 - val_accuracy: 0.000(
Epoch 24/100
10/10 [==============================] - 0s 4ms/step - loss: 3360.0625 - accuracy: 0.0000e+00 - val_loss: 3305.3931 - val_accuracy: 0.000(
Epoch 25/100
10/10 [==============================] - 0s 4ms/step - loss: 3356.1685 - accuracy: 0.0000e+00 - val_loss: 3301.4180 - val_accuracy: 0.000(
Epoch 26/100
10/10 [==============================] - 0s 3ms/step - loss: 3352.3015 - accuracy: 0.0000e+00 - val_loss: 3297.4368 - val_accuracy: 0.000(
Epoch 27/100
10/10 [==============================] - 0s 4ms/step - loss: 3348.2852 - accuracy: 0.0000e+00 - val_loss: 3293.5952 - val_accuracy: 0.000(
Epoch 28/100
10/10 [==============================] - 0s 3ms/step - loss: 3344.4700 - accuracy: 0.0000e+00 - val_loss: 3289.6953 - val_accuracy: 0.000(
Epoch 29/100
10/10 [==============================] - 0s 3ms/step - loss: 3340.6382 - accuracy: 0.0000e+00 - val_loss: 3285.7625 - val_accuracy: 0.000(
```

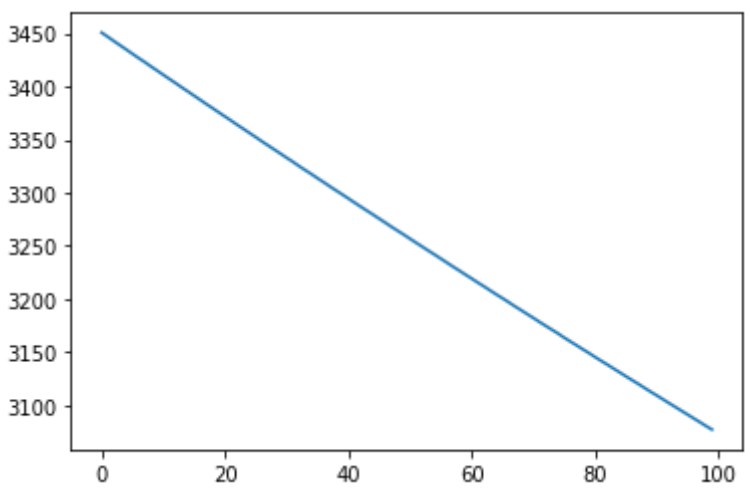
## (4) Model Predict

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

## (5) 학습 결과 시각화

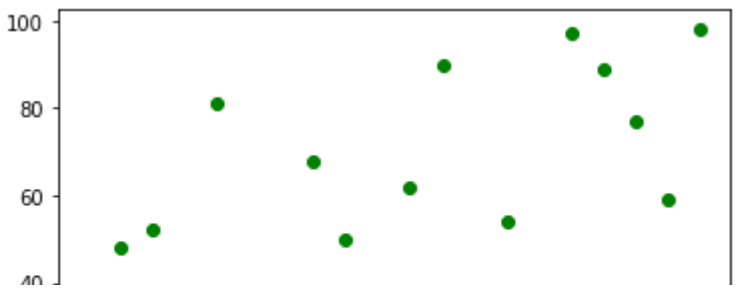
- Loss 감소

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




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

```
plt.scatter(range(20), y_test, c = 'g')
plt.scatter(range(20), y_hat, c = 'r')
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 21ms/step - loss: 0.2941 - accuracy: 0.0125 - val_loss: 0.2763 - val_accuracy: 0.0000e+00
Epoch 2/100
10/10 [==============================] - 0s 3ms/step - loss: 0.2730 - accuracy: 0.0125 - val_loss: 0.2545 - val_accuracy: 0.0000e+00
Epoch 3/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2520 - accuracy: 0.0125 - val_loss: 0.2333 - val_accuracy: 0.0000e+00
Epoch 4/100
10/10 [==============================] - 0s 3ms/step - loss: 0.2309 - accuracy: 0.0125 - val_loss: 0.2126 - val_accuracy: 0.0000e+00
Epoch 5/100
10/10 [==============================] - 0s 3ms/step - loss: 0.2104 - accuracy: 0.0125 - val_loss: 0.1919 - val_accuracy: 0.0000e+00
Epoch 6/100
10/10 [==============================] - 0s 3ms/step - loss: 0.1907 - accuracy: 0.0125 - val_loss: 0.1707 - val_accuracy: 0.0000e+00
Epoch 7/100
10/10 [==============================] - 0s 3ms/step - loss: 0.1683 - accuracy: 0.0125 - val_loss: 0.1514 - val_accuracy: 0.0000e+00
Epoch 8/100
10/10 [==============================] - 0s 3ms/step - loss: 0.1492 - accuracy: 0.0125 - val_loss: 0.1315 - val_accuracy: 0.0000e+00
Epoch 9/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1301 - accuracy: 0.0125 - val_loss: 0.1124 - val_accuracy: 0.0000e+00
Epoch 10/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1116 - accuracy: 0.0125 - val_loss: 0.0953 - val_accuracy: 0.0000e+00
Epoch 11/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0951 - accuracy: 0.0125 - val_loss: 0.0806 - val_accuracy: 0.0000e+00
Epoch 12/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0823 - accuracy: 0.0125 - val_loss: 0.0676 - val_accuracy: 0.0000e+00
Epoch 13/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0702 - accuracy: 0.0125 - val_loss: 0.0582 - val_accuracy: 0.0000e+00
Epoch 14/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0626 - accuracy: 0.0125 - val_loss: 0.0507 - val_accuracy: 0.0000e+00
Epoch 15/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0563 - accuracy: 0.0250 - val_loss: 0.0456 - val_accuracy: 0.0000e+00
Epoch 16/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0514 - accuracy: 0.0250 - val_loss: 0.0423 - val_accuracy: 0.0000e+00
Epoch 17/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0481 - accuracy: 0.0250 - val_loss: 0.0399 - val_accuracy: 0.0000e+00
Epoch 18/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0456 - accuracy: 0.0250 - val_loss: 0.0379 - val_accuracy: 0.0000e+00
Epoch 19/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0437 - accuracy: 0.0250 - val_loss: 0.0360 - val_accuracy: 0.0000e+00
Epoch 20/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0415 - accuracy: 0.0250 - val_loss: 0.0343 - val_accuracy: 0.0000e+00
Epoch 21/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0394 - accuracy: 0.0250 - val_loss: 0.0326 - val_accuracy: 0.0000e+00
Epoch 22/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0374 - accuracy: 0.0250 - val_loss: 0.0310 - val_accuracy: 0.0000e+00
Epoch 23/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0355 - accuracy: 0.0250 - val_loss: 0.0293 - val_accuracy: 0.0000e+00
Epoch 24/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0337 - accuracy: 0.0250 - val_loss: 0.0277 - val_accuracy: 0.0000e+00
Epoch 25/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0316 - accuracy: 0.0250 - val_loss: 0.0261 - val_accuracy: 0.0000e+00
Epoch 26/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0297 - accuracy: 0.0250 - val_loss: 0.0245 - val_accuracy: 0.0000e+00
Epoch 27/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0277 - accuracy: 0.0250 - val_loss: 0.0230 - val_accuracy: 0.0000e+00
Epoch 28/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0259 - accuracy: 0.0250 - val_loss: 0.0214 - val_accuracy: 0.0000e+00
Epoch 29/100
10/10 [==============================] - 0s 3ms/step - loss: 0.0240 - accuracy: 0.0250 - val_loss: 0.0199 - val_accuracy: 0.0000e+00
Epoch 30/100
```

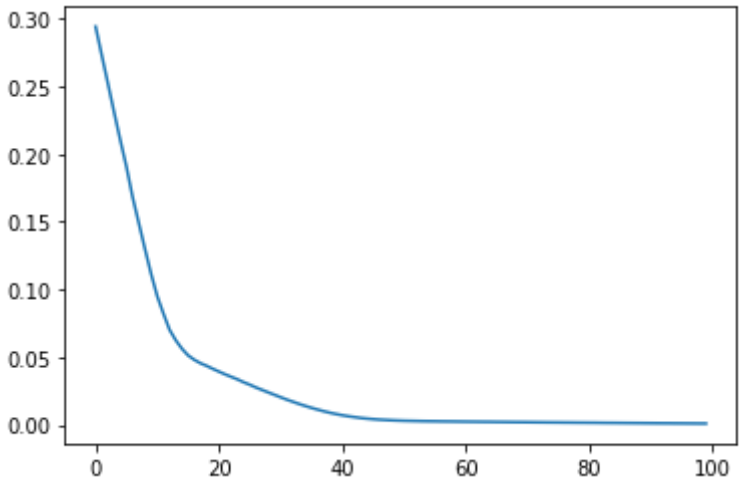
## (4) Model Predict

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

### (5) 학습 결과 시각화

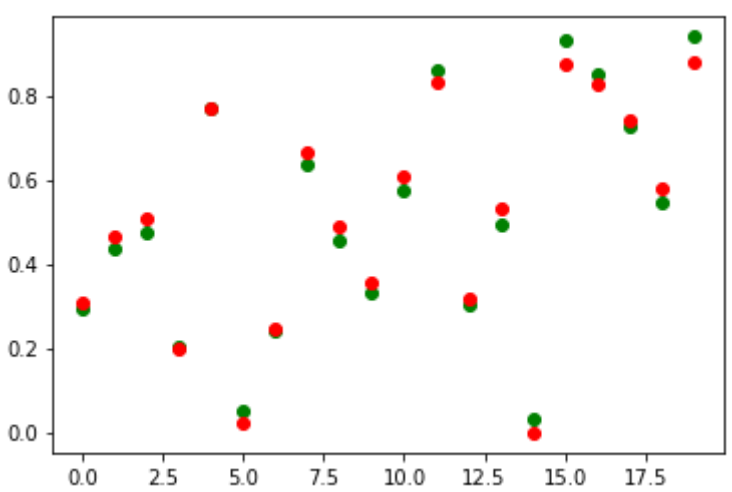
- Loss 감소

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




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

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




## III. Stacked_SimpleRNN

### 1) Model Define & Summary

- return_sequences = True

```
model_3 = models.Sequential(name = 'Stackd_RNN')
model_3.add(layers.SimpleRNN(3,
                             input_shape = (None, 1),
                             return_sequences = True))
model_3.add(layers.SimpleRNN(3,
                             input_shape = (None, 1),
                             return_sequences = False))
model_3.add(layers.Dense(1))

model_3.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_3.compile(loss = 'mse',
                optimizer = 'adam',
                metrics = ['accuracy'])
```

## 3) Model Fit

```
Hist_3 = model_3.fit(X_train, y_train,
                     epochs = 100,
                     batch_size = 8,
                     validation_data = (X_test, y_test))
```

```
Epoch 1/100
10/10 [==============================] - 1s 29ms/step - loss: 0.1946 - accuracy: 0.0250 - val_loss: 0.1532 - val_accuracy: 0.0000e+00
Epoch 2/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1246 - accuracy: 0.0250 - val_loss: 0.0985 - val_accuracy: 0.0000e+00
Epoch 3/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0774 - accuracy: 0.0250 - val_loss: 0.0635 - val_accuracy: 0.0000e+00
Epoch 4/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0518 - accuracy: 0.0250 - val_loss: 0.0456 - val_accuracy: 0.0000e+00
Epoch 5/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0400 - accuracy: 0.0250 - val_loss: 0.0388 - val_accuracy: 0.0000e+00
Epoch 6/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0359 - accuracy: 0.0250 - val_loss: 0.0365 - val_accuracy: 0.0000e+00
Epoch 7/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0346 - accuracy: 0.0250 - val_loss: 0.0347 - val_accuracy: 0.0000e+00
Epoch 8/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0330 - accuracy: 0.0250 - val_loss: 0.0331 - val_accuracy: 0.0000e+00
Epoch 9/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0313 - accuracy: 0.0250 - val_loss: 0.0315 - val_accuracy: 0.0000e+00
Epoch 10/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0297 - accuracy: 0.0250 - val_loss: 0.0301 - val_accuracy: 0.0000e+00
Epoch 11/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0282 - accuracy: 0.0250 - val_loss: 0.0285 - val_accuracy: 0.0000e+00
Epoch 12/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0265 - accuracy: 0.0250 - val_loss: 0.0271 - val_accuracy: 0.0000e+00
Epoch 13/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0250 - accuracy: 0.0250 - val_loss: 0.0257 - val_accuracy: 0.0000e+00
Epoch 14/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0236 - accuracy: 0.0250 - val_loss: 0.0242 - val_accuracy: 0.0000e+00
Epoch 15/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0224 - accuracy: 0.0250 - val_loss: 0.0230 - val_accuracy: 0.0000e+00
Epoch 16/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0209 - accuracy: 0.0250 - val_loss: 0.0215 - val_accuracy: 0.0000e+00
Epoch 17/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0196 - accuracy: 0.0250 - val_loss: 0.0201 - val_accuracy: 0.0000e+00
Epoch 18/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0184 - accuracy: 0.0250 - val_loss: 0.0189 - val_accuracy: 0.0000e+00
Epoch 19/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0173 - accuracy: 0.0250 - val_loss: 0.0177 - val_accuracy: 0.0000e+00
Epoch 20/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0162 - accuracy: 0.0250 - val_loss: 0.0165 - val_accuracy: 0.0000e+00
Epoch 21/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0152 - accuracy: 0.0250 - val_loss: 0.0158 - val_accuracy: 0.0000e+00
Epoch 22/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0143 - accuracy: 0.0250 - val_loss: 0.0144 - val_accuracy: 0.0000e+00
Epoch 23/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0132 - accuracy: 0.0250 - val_loss: 0.0136 - val_accuracy: 0.0000e+00
Epoch 24/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0123 - accuracy: 0.0250 - val_loss: 0.0126 - val_accuracy: 0.0000e+00
Epoch 25/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0115 - accuracy: 0.0250 - val_loss: 0.0118 - val_accuracy: 0.0000e+00
Epoch 26/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0106 - accuracy: 0.0250 - val_loss: 0.0109 - val_accuracy: 0.0000e+00
Epoch 27/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0100 - accuracy: 0.0250 - val_loss: 0.0100 - val_accuracy: 0.0000e+00
Epoch 28/100
```

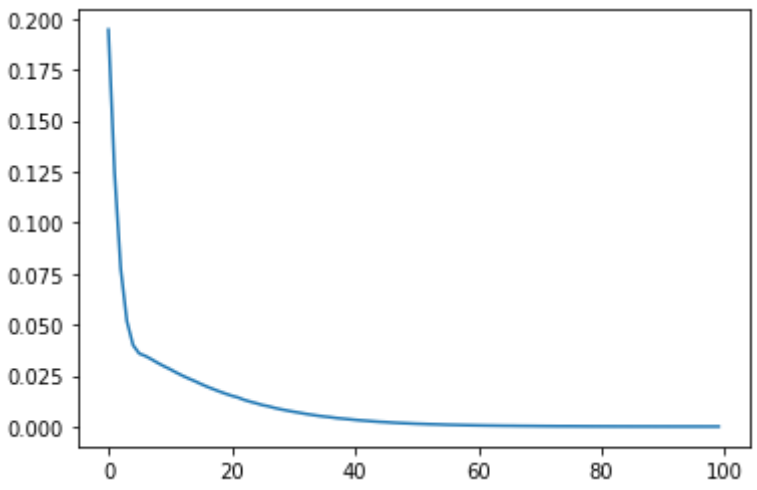
```
10/10 [==============================] - 0s 4ms/step - loss: 0.0092 - accuracy: 0.0250 - val_loss: 0.0095 - val_accuracy: 0.0000e+00
Epoch 29/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0085 - accuracy: 0.0250 - val_loss: 0.0087 - val_accuracy: 0.0000e+00
```

### 4) Model Predict

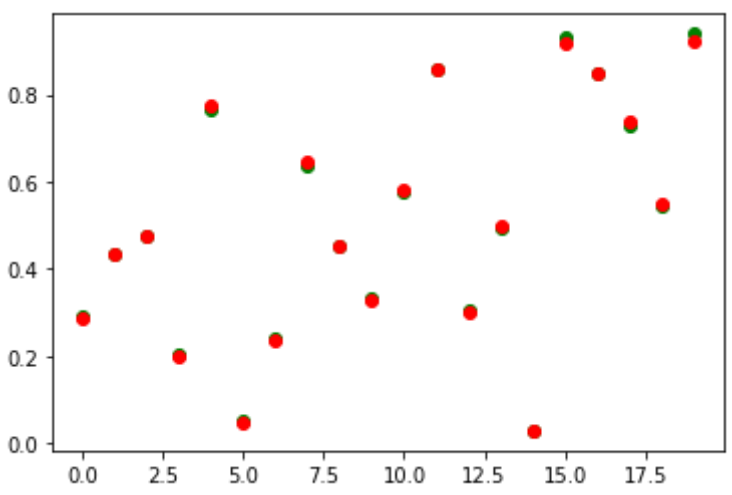
```
y_hat = model_3.predict(X_test)
```

### 5) 학습 결과 시각화

```
plt.plot(Hist_3.history['loss'])
plt.show()
```




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

```
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))
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fa7bad84710> triggerec
array([[-0.877573,  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'])
```

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

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fa7bae13560> triggerec
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.        , -0.99998426,  1.        ],
        [ 1.        , -0.9998903 ,  1.        ],
        [ 1.        , -0.9999269 ,  1.        ],
        [ 1.        , -0.9999513 ,  1.        ],
        [ 1.        , -0.99996746,  1.        ]]], dtype=float32)
```

#

#

#

# The End

#

#

#