넘파이로 구현한 간단한 RNN

In [5]:

```
import keras
keras.__version__
```

Out[5]:

```
'2.4.3'
```

In [6]:

```
import numpy as np
```

In [7]:

```
timesteps = 100         # 입력 시퀀스에 있는 타임스텝의 수
input_features = 32     # 입력 특성의 차원
output_features = 64    # 출력 특성의 차원
```

In [8]:

```
inputs = np.random.random( (timesteps, input_features)  )
print( inputs.shape )
inputs
```

```
(100, 32)
```

Out[8]:

```
array([[0.04660541, 0.16109852, 0.28930481, ..., 0.58336442, 0.98320904,
        0.25654855],
       [0.27512999, 0.46278217, 0.6902794 , ..., 0.1095859 , 0.37062854,
        0.33986645],
       [0.40936041, 0.08661058, 0.20617565, ..., 0.26904209, 0.65805157,
        0.39939449],
       ...,
       [0.09060803, 0.93794775, 0.04682485, ..., 0.08070006, 0.7308984 ,
        0.70537931],
       [0.65547604, 0.11648888, 0.57752614, ..., 0.56265962, 0.38423102,
        0.37779771],
       [0.30472319, 0.56550675, 0.13771781, ..., 0.29715877, 0.90835744,
        0.89162959]])
```

In [9]:

```
### 초기상태 : 모두 0인 벡터
state_t = np.zeros( (output_features, ))
print( state_t.shape )
state_t
```

(64,)

Out[9]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

랜덤한 가중치 행렬

In [10]:

```
W = np.random.random( (output_features, input_features ))
U = np.random.random( (output_features, output_features ))
b = np.random.random( (output_features,  ))

print(W.shape, U.shape, b.shape)
print(W[0], end='₩n₩n')
print(U[0], end='₩n₩n')
print(b[0], end='₩n₩n')
```

```
(64, 32) (64, 64) (64,)
[0.63094667 0.5723013  0.81277864 0.04122926 0.14264744 0.70868253
 0.04141886 0.08370889 0.67300665 0.882774   0.19456693 0.92237371
 0.74105043 0.87864024 0.83264715 0.75150268 0.86428701 0.71381844
 0.57606533 0.36161968 0.47746347 0.28258501 0.1500741  0.72156362
 0.72938387 0.94381753 0.07889748 0.98200566 0.8065512  0.93703383
 0.27975107 0.37509746]

[0.91983942 0.54960007 0.37028701 0.73127441 0.34519301 0.17428552
 0.88029849 0.02418245 0.34185041 0.98075607 0.1341119  0.89111246
 0.5120043  0.14706754 0.61786284 0.40877601 0.80136436 0.23554382
 0.76599093 0.73620383 0.60698166 0.20202944 0.9021647  0.41605377
 0.85303496 0.89827042 0.43149938 0.038201   0.21021936 0.23103071
 0.75828039 0.8948224  0.3709417  0.54689727 0.88407145 0.71921952
 0.1227101  0.53025138 0.70389796 0.50343851 0.78847769 0.27000007
 0.92797701 0.75404272 0.37027352 0.69946837 0.97544953 0.86608364
 0.23732397 0.0946843  0.80752907 0.90355454 0.84685838 0.97336459
 0.2044802  0.12075474 0.75180145 0.73890759 0.69511194 0.41082873
 0.00241194 0.88755926 0.79954811 0.25813738]

0.7843628623440486
```

연속된 결과

In [11]:

```
# inputs = np.random.random( (timesteps, input_features)  )  100, 32
suc_outputs = []
for input_t in inputs:
    print(input_t.shape, input_t)

    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
    suc_outputs.append(output_t)
    state_t = output_t

final_output_sequence = np.stack(suc_outputs, axis=0)
```

```
(32,) [0.92977578 0.60346617 0.11413642 0.70850363 0.25684481 0.52364254
 0.80874139 0.5930605  0.70710916 0.78638454 0.71197628 0.96260308
 0.06548135 0.7601525  0.56516692 0.25753396 0.19186053 0.92923327
 0.90131832 0.53294697 0.41444626 0.98964812 0.12793823 0.70797688
 0.61946015 0.88714065 0.95418434 0.72425595 0.64539257 0.26405187
 0.70608996 0.61053075]
(32,) [0.38953431 0.54379416 0.281859   0.59563928 0.75342985 0.32822917
 0.12927119 0.90721323 0.81337283 0.48257704 0.15643618 0.5623927
 0.86864633 0.01339686 0.41267523 0.89591024 0.40544164 0.83086053
 0.39980414 0.36680765 0.83017066 0.70745184 0.60598517 0.79905305
 0.51879837 0.65950042 0.16894111 0.640829   0.09657262 0.32308126
 0.31433268 0.19778125]
(32,) [0.24450566 0.60807474 0.05343623 0.84579917 0.92876186 0.92544566
 0.24709248 0.92645644 0.55231564 0.38786457 0.04224035 0.43202383
 0.52772967 0.37203785 0.06142665 0.83144398 0.95525275 0.19541631
 0.93473273 0.02719006 0.37697259 0.06559588 0.35689175 0.90747414
 0.58145518 0.16227678 0.39637308 0.59459594 0.09867088 0.30499669
 0.41994754 0.17880731]
(32,) [0.73617732 0.83470203 0.99466365 0.11520323 0.42608505 0.76747007
 0.14845391 0.07199072 0.39915827 0.28166135 0.69513916 0.4431319
```

In [12]:

```
final_output_sequence.shape
```

Out[12]:

```
(100, 64)
```

In [13]:

```
final_output_sequence[0][0:32]
```

Out[13]:

```
array([0.99999999, 0.99999982, 1.        , 0.99999999, 0.99999993,
       0.99999981, 0.99999998, 0.99999994, 0.99999991, 0.99999994,
       0.99999999, 0.99999999, 0.99999999, 0.99999999, 1.        ,
       0.9999999 , 0.99999996, 1.        , 0.99999996, 0.99999998,
       0.99999988, 0.99999996, 0.99999999, 0.9999993 , 0.99999993,
       0.99999998, 1.        , 0.99999976, 0.99999996, 0.99999999,
       1.        , 0.99999978])
```

02 케라스를 활용한 RNN 구현

- 입력 (batch_size, timesteps, input_features)
  - batch_size : 배치 사이즈
  - timesteps : 시간 수준
  - input_features : 입력 차원수

SimpleRNN의 두가지 모드

- (1) 각 타임스텝의 출력을 모은 전체 시퀀스를 반환 (batch_size, timesteps, output_feature) 3D텐서
- (2) 입력 시퀀스에 대한 마지막 출력만 반환(batch_size, output_features) 2D텐서
- return_sequences의 매개변수로 선택이 가능.

(2) 입력 시퀀스에 대한 마지막 출력만 반환

- (batch_size, output_features) 2D텐서

In [17]:

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN

model = Sequential()
model.add(Embedding(10000, 32))  # 시퀀스 길이 10000, 차원 32로
model.add(SimpleRNN(32, return_sequences=False))  #  return_sequences는 기본값이 False
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

(1) 출력을 모은 전체 시퀀스 반환

- (batch_size, timesteps, output_feature) 3D텐서

In [18]:

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, None, 32)          2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

여러개의 순환 층을 차례대로 쌓아보기

- 이런 설정에서는 중간 층들이 전체 출력 시퀀스를 반환하도록 설정해야 한다.

In [20]:

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32))  # 맨 위 층만 마지막 출력을 반환합니다.
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_4 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_5 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
_________________________________________________________________
```

In [ ]: