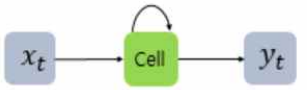
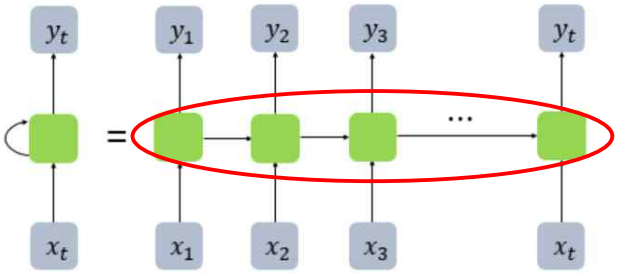
순환 신경망은 딥러닝에 있어 가장 기본적은 Sequence 모델이다. RNN(Recurrent Neural Network)라고 불리우며 입력과 출력을 시퀀스 단위로 처리하는 모델이다.

RNN을 처음 배우게 되면 아래 이미지로 소개를 많이 할 것이다.



하지만 이해를 돕기 위해 아래 이미지로 기본적인 RNN의 구조에 대해 이해할 것이다.

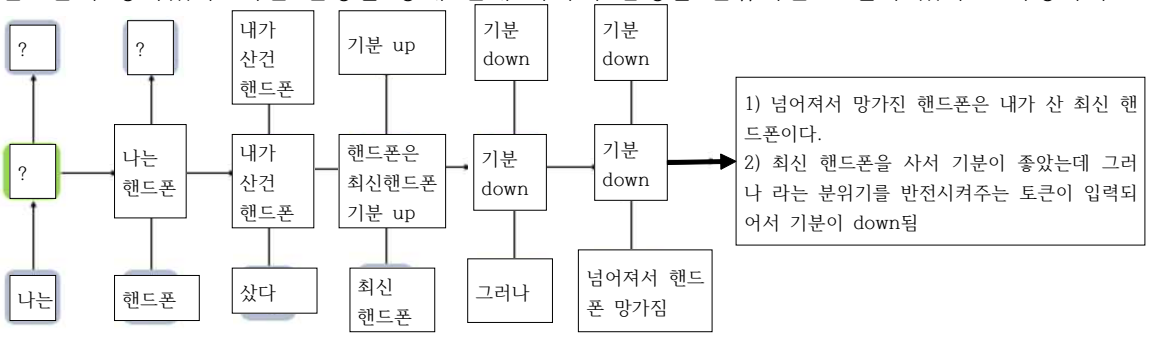


x_t는 input data로 $x_t = (x_1, x_2, x_3 \ \ldots x_t)$로 구성되어 있다. x_t를 Sequence data라고 부르고 $Len(x_t)$를 Sequence Length라고 부른다.

y_t는 input data로 many to many, many to one, one to many 형식에 따라서 Sequence Length만큼 출력이 나올수도, 1개만 나올수도 있다.

빨간색 원을 메모리 셀 또는 RNN 셀이라고 부른다. 이것은 입력에 있어서 1개의 Sequence data에 매핑되는 출력이며 곧 다음 은닉층으로 데이터를 넘겨주는 역할을 한다.

예를들어 “나는 오늘 핸드폰을 샀는데 그 핸드폰은 정말 최신형 핸드폰이다. 그러나 넘어져서 핸드폰이 망가졌다.”라는 문장을 통해 현재 화자의 감정을 분류하는 모델이 있다고 가정하자.



위와 같은 순서로 입력되어 과거의 입력된 데이터와 현재 입력된 데이터를 기반으로 화자의 기분을 분류할 수 있다.

실제 모델이 저렇게 동작한다는 것은 아니고 RNN은 저런 느낌으로 동작한다 정도만 이해하면 될 것이다.

만약 Many to many로 RNN 모델을 설계했다면 “감정의 변화”를 알 수 있고, Many to one 모델을 설계했다면 “현재 감정 상태”를 알 수 있다.

그 외 스팸메일 분류, 개체명 인식, 품사 태깅, 번역기 등 필요에 따라 Many to many, May to one 등을 설정해 모델을 설계할 필요가 있다.

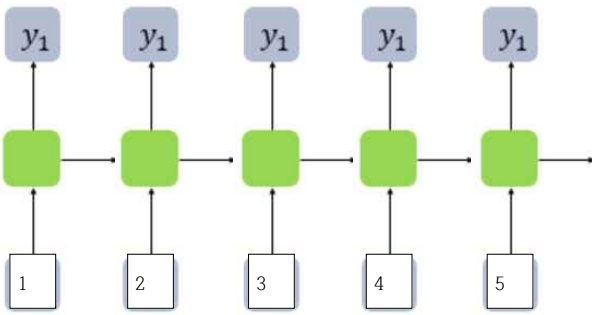
$$\text{은닉층} : h_t = tanh(W_x x_t + W_h h_{t-1} + b)$$
$$\text{출력층} : y_t = f(W_y h_t + b)$$

위 사진은 RNN에 대한 수식을 정리한 것이다. 실제 위와 같이 동작하는지 아래 코드로 확인해 보도록 한다.

Layer들 Weight와 Bias를 직접 찍어보기 위해 Subclassing model을 사용하였다.

```
from typing import Sequence
import numpy as np
from tensorflow.keras.layers import SimpleRNN, Layer, Dense
from tensorflow.keras.models import Model, Sequential

class TestModel(Model) :
    def __init__(self) :
        super(TestModel, self).__init__()

        self.dense1 = SimpleRNN(1, input_shape=(5,1), return_sequences=True) # RNN Layer


    def call(self, x): # forward propagation
        x = self.dense1(x) # forward propagation RNN Layer
        input_matrix, recurent_matrix, bias_matrix = self.dense1.get_weights() # Get weights
        print("input_matrix : \n",input_matrix)
        print("recurent_matrix : \n",recurent_matrix)
        print("bias_matrix : \n",bias_matrix)

        return x

model_class = TestModel()

train_x = np.array([[[1],[2],[3],[4],[5]]], dtype=np.float)
print(train_x.shape)

Y_Class = model_class(train_x)
print(Y_Class)
```

```
input_matrix :
 [[0.32394445]]
recurent_matrix :
 [[1.]]
bias_matrix :
 [0.]
tf.Tensor(
[[[0.31306913]
  [0.7447039 ]
  [0.9374446 ]
  [0.97728485]
  [0.98896194]]], shape=(1, 5, 1), dtype=float32)
```



$W_x = 0.3239, W_h = 1, Bias = 0$

위 값들을 manual로 계산한 식을 아래에 첨부한다.

```python
import numpy as np


input = [1,2,3,4,5]
W_x = 0.3239
W_h = 1
Bias = 0

y_1 = np.tanh(input[0]*W_x)
y_2 = np.tanh(input[1]*W_x+y_1*W_h)
y_3 = np.tanh(input[2]*W_x+y_2*W_h)
y_4 = np.tanh(input[3]*W_x+y_3*W_h)
y_5 = np.tanh(input[4]*W_x+y_4*W_h)

print(y_1)
print(y_2)
print(y_3)
print(y_4)
print(y_5)
```

```
0.31302905318728924
0.7446464400206215
0.9374215851351907
0.9772757615483589
0.988956808981887
```

```
tf.Tensor(
[[[0.31306913]
  [0.7447039 ]
  [0.9374446 ]
  [0.97728485]
  [0.98896194]]], shape=(1, 5, 1), dtype=float32)
```

forward propagation을 manual로 계산한 결과와 tensorflow가 계산한 결과가 거의 동일한 것을 확인할 수 있다.