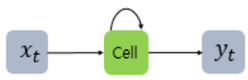
RNN

RNN은?

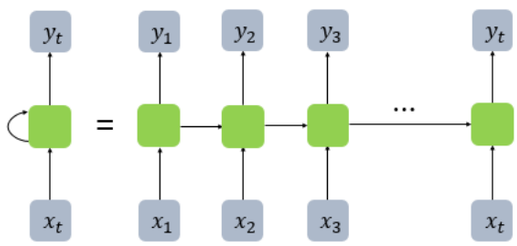
RNN (Recurrent Neural Network) - 순환신경망

- sequence 모델 : 입/출력을 시퀀스 단위로 처리 → 맥락/순차가 있는 series 데이터를 사용
- 은닉층에서 활성화 함수를 지닌 값이 출력층 방향으로 가는 Feed Forward Neural Network와 달리 은닉층에서 활성화 함수를 지닌 값이 출력층으로도 가고, 다음 은닉층의 입력으로도 가는 특징을 가진 RNN → 순환

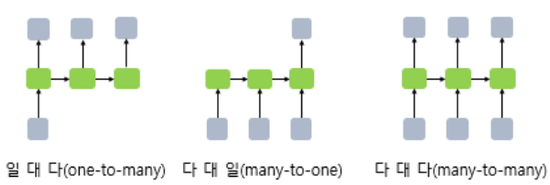


은닉층에서 활성화함수를 통해 결과를 출력하는 역할 : cell (=RNN cell, 메모리 cell)→ 이전 값을 기억하는 메모리 역할

⇒ cell은 각 시점 (time step)에서 이전 시점의 메모리 셀에서 나온 값인 은닉 상태 (hidden state)를 자신의 입력으로 사용하는 재귀적 활동을 함



- RNN은 입출력 길이를 다르게 설계 할 수 있음
  1. one-to-many : image captioning에 사용 (한 이미지 입력에 대해 사진의 제목 출력)
  2. many-to-one : 감성 분류, 스팸 메일 분류에 사용
  3. many-to-many : 챗봇, 번역기 등에 사용



RNN 수식

현재 시점 t에서의 은닉 상태값을 ht라 할 때, 메모리 셀이 ht를 계산하기 위해 총 두 개 가중치가 필요 - Wx, Wh

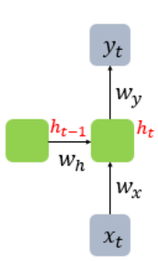
Wx : 입력층에서 입력값을 위한 가중치

Wh : 이전 시점 t-1의 은닉 상태값인 ht-1을 위한 가중치

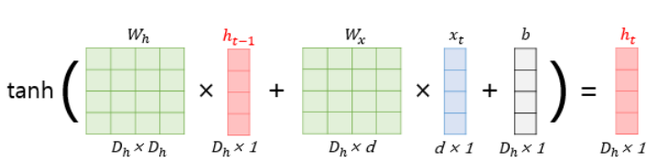
$$\text{은닉층} : h_t = tanh(W_x x_t + W_h h_{t-1} + b)$$
$$\text{출력층} : y_t = f(W_y h_t + b)$$

활성화 함수 F : 하이퍼볼릭탄젠트(비선형 함수)

Wh : 출력층에서 출력값을 위한 가중치



단어 벡터의 차원이 d, 은닉 상태의 크기가 Dh라 했을 때, 각 벡터의 행렬 크기 :



RNN의 한계

단순한 형태의 RNN을 바닐라 RNN이라고 함

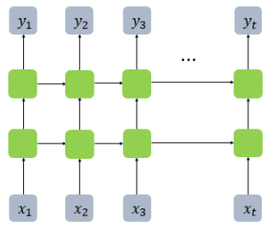
바닐라 RNN의 단점 : 장기 의존성 문제 = long-term dependency

⇒ time step이 길어질 수록 앞의 정보가 충분히 전달되지 못함

⇒ 즉 문장이 길 수록 단어를 예측하기 어려움

Deep Recurrent Neural Network (깊은 순환 신경망)

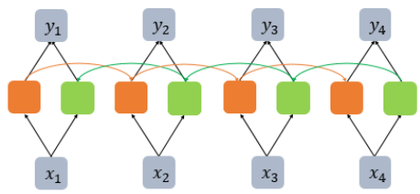
- 다수의 은닉층을 가짐



은닉층이 2개

Bidirectional Recurrent Neural Network (양방향 순환 신경망)

- 이전 시점의 메모리 셀뿐만 아니라 이후의 메모리 셀로도 예측할 수 있음
- 아래 그림의 주황색 메모리 셀이 앞 시점의 은닉 상태, 초록색 메모리 셀이 뒤 시점의 은닉 상태를 전달 받음



파이썬으로 RNN 구현

```python
import numpy as np

timesteps = 10 # input의 길이
input_dim = 4 # 단어 벡터 차원
hidden_size = 8 # 메모리 셀 용량

inputs = np.random.random((timesteps, input_dim))
hidden_state_t = np.zeros((hidden_size))

Wx = np.random.random((hidden_size, input_dim))
Wh = np.random.random((hidden_size, hidden_size))
b = np.random.random((hidden_size))

total_hidden_state = []

for input_t in inputs:
    output_t = np.tanh(np.dot(Wx, input_t) + np.dot(Wh, hidden_state_t) + b)
    total_hidden_state.append(output_t)
    hidden_state_t = output_t

total_hidden_states = np.stack(total_hidden_state, axis = 0)
print(total_hidden_states)
```

출력

```
[[0.83213084 0.94014708 0.94293618 0.9482776  0.95779251 0.92466643
  0.96846002 0.89850639]
 [0.99987603 0.99998275 0.99969981 0.99997101 0.99999835 0.99998026
  0.99999751 0.99988123]
 [0.99990898 0.99999164 0.99963683 0.99996511 0.99999688 0.99998957
  0.99999931 0.99977258]
 [0.99984286 0.99997693 0.99917386 0.99989104 0.99999534 0.99997721
  0.99999673 0.99971713]
 [0.99992776 0.99998457 0.99962919 0.99997349 0.99999718 0.99999537
  0.99999898 0.9998552 ]
 [0.9999532  0.99998916 0.99971282 0.99998716 0.99999909 0.99999549
  0.99999896 0.99990327]
 [0.99994704 0.9999913  0.99987952 0.99999177 0.99999922 0.99999642
  0.99999942 0.99995932]
 [0.99985877 0.99997671 0.99938228 0.99992068 0.99999566 0.99998501
  0.99999743 0.99979908]
 [0.99978089 0.99996169 0.99931316 0.9998622  0.99999456 0.99997645
  0.99999499 0.99984046]
 [0.99992143 0.99999046 0.99952873 0.999966   0.99999789 0.99998829
  0.9999989  0.9997608 ]]
```