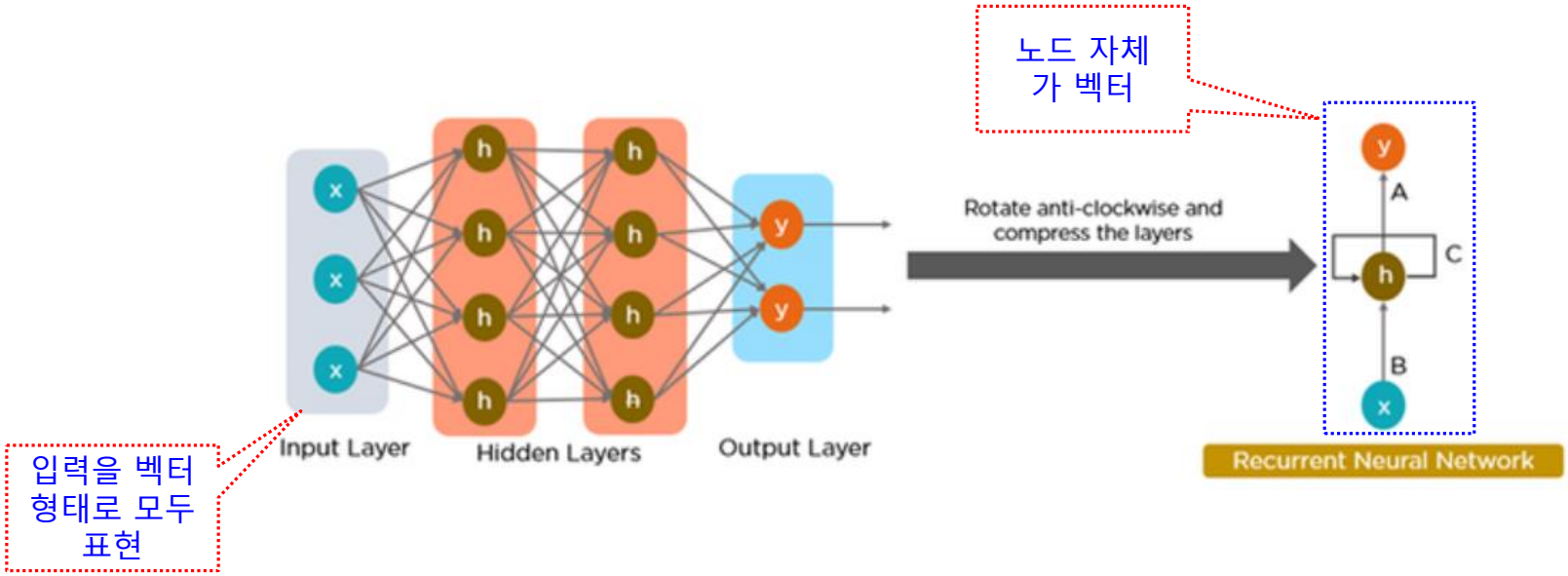
순환신경망 RNN

7장 순환신경망

2020.08.13목 2h

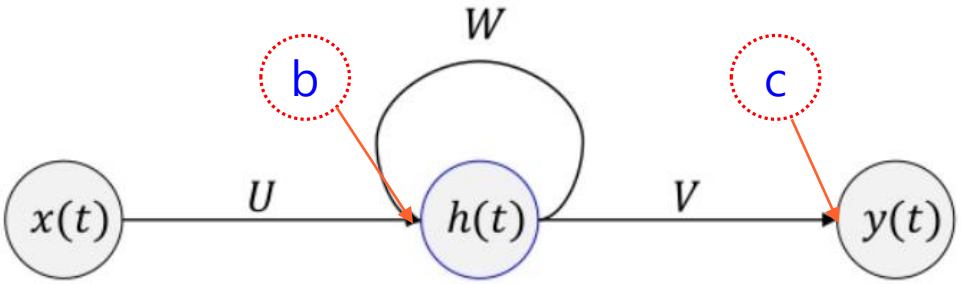
순환 신경망 개요

- **Recurrent Neural Networks(RNN)**
  - 순서가 있는 데이터를 입력으로 받고, 같은 네트워크를 이용해 변화하는 입력에 대한 출력을 생성
    - **시간과 연관된 일**
      - **예를 들자면, 책을 읽을 때 앞에 있는 내용을 기억하면서 글을 읽음**
  - 순서가 있는 데이터
    - **음악, 자연어, 날씨, 주가 등 시간의 흐름에 따라 변화하고 그 변화가 의미를 갖는 데이터**



RNN(Recurrent Neural Network) 기본 구조

- **첫 번째 활성화 함수**
  - 보통 tanh(x) 사용
  - 어떤 t(타임스텝)에 대해서도 같은 가중치 U, W, V, 편향 b, c를 사용
    - **가중치와 편향을 공유**
      - **계산량이 감소**
    - **편향은 언급하지 않는 경우도 대부분**



$$h(t) = f(Ux(t) + Wh(t-1) + b)$$

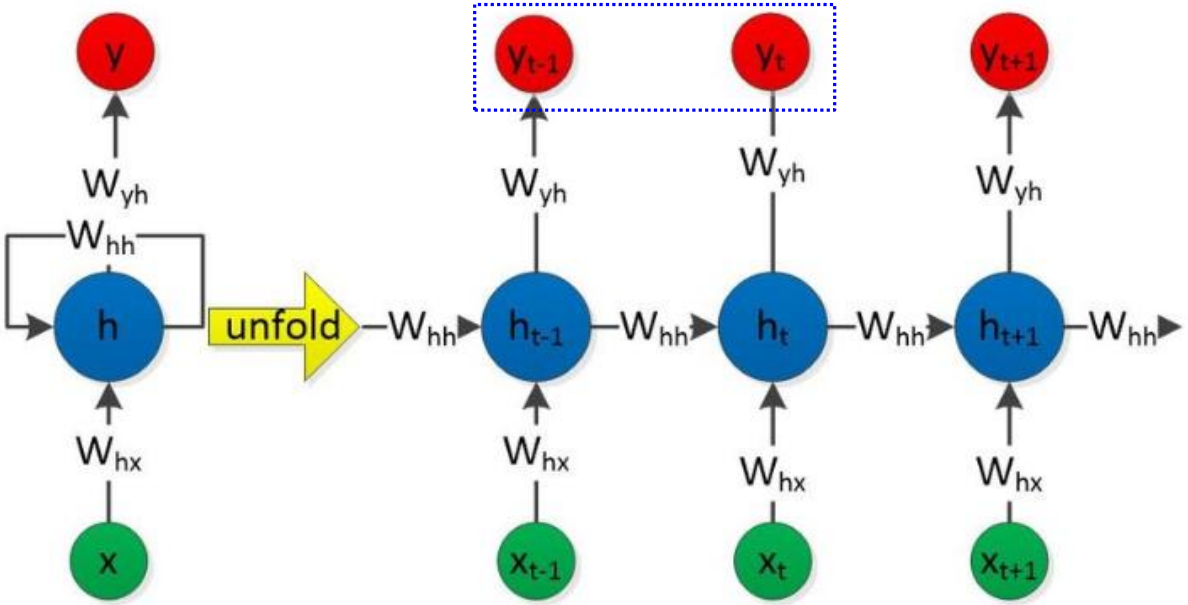
$$y(t) = g(Vh(t) + c)$$

직관적인 표현

* $x(t)$: **현재 입력 (자극)**
* $h(t\text{-}1)$: **과거 기억**
* $h(t)$: **현재 기억**

RNN 딥러닝 구조

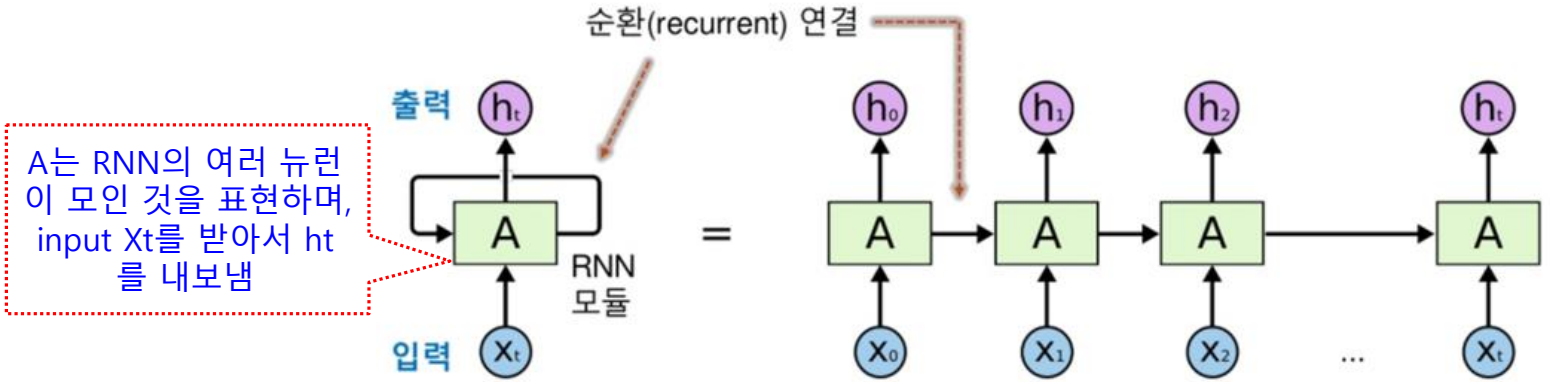
- **순환 모양의 화살표**
  - 중간층이 순환
    - **순환의 각각을 타임스텝 t-1, t, t+1이라 함**
- **입력 X를 받아서, 출력 Y를 반환하는 순환구조**
  - 어떤 레이어의 출력을 다시 입력으로 받는 구조
  - 순환 신경망은 입력과 출력의 길이에 무제한

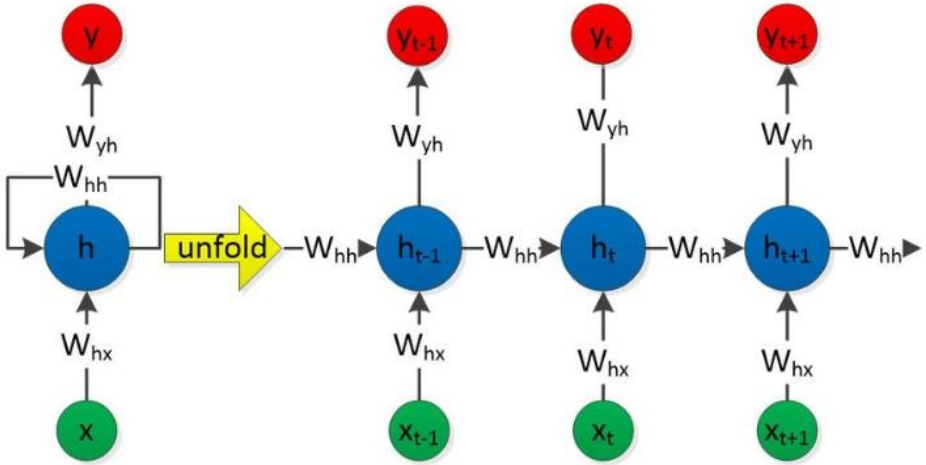


The architecture of RNN.

RNN 딥러닝 구조의 다양한 그림

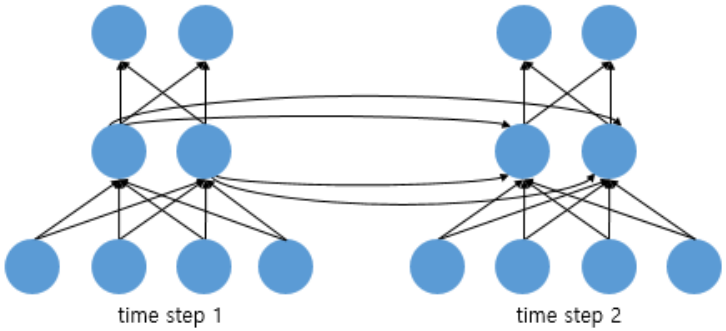
- **입력 X를 받아서, 출력 Y를 반환하는 순환구조**
  - 순환 모양의 화살표, 중간층이 순환





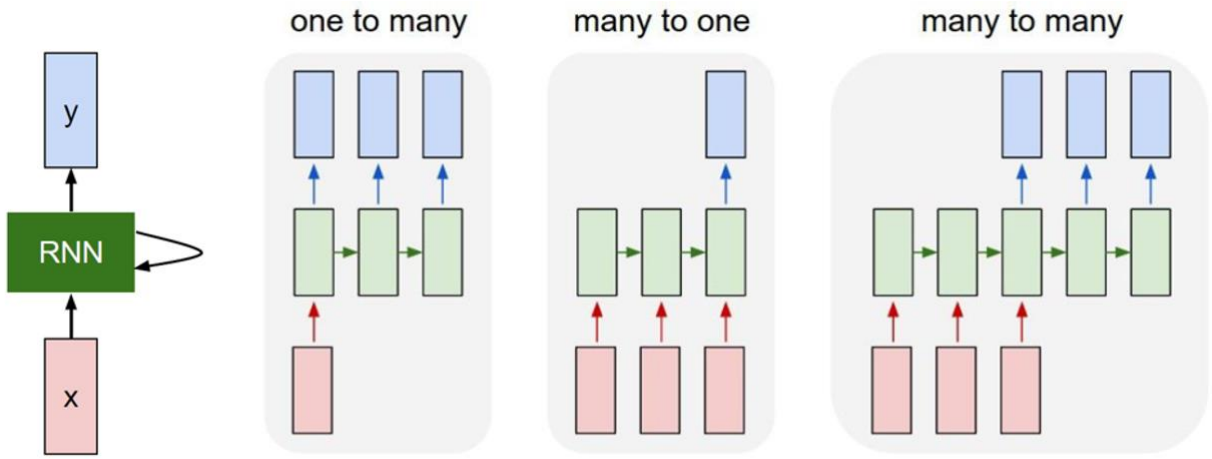
The architecture of RNN.

뉴런 단위로 표현



RNN 활용

- **활용 분야**
  - 이미지 설명 문구 생성
    - **순환 신경망은 이미지에 대한 설명을 생성**
  - 문장 감성 분석
    - **문장의 긍정/부정을 판단하는 감성 분석**
  - 기계 번역(Machine Translation) 등
    - **하나의 언어를 다른 언어로 번역**



<기본구조>

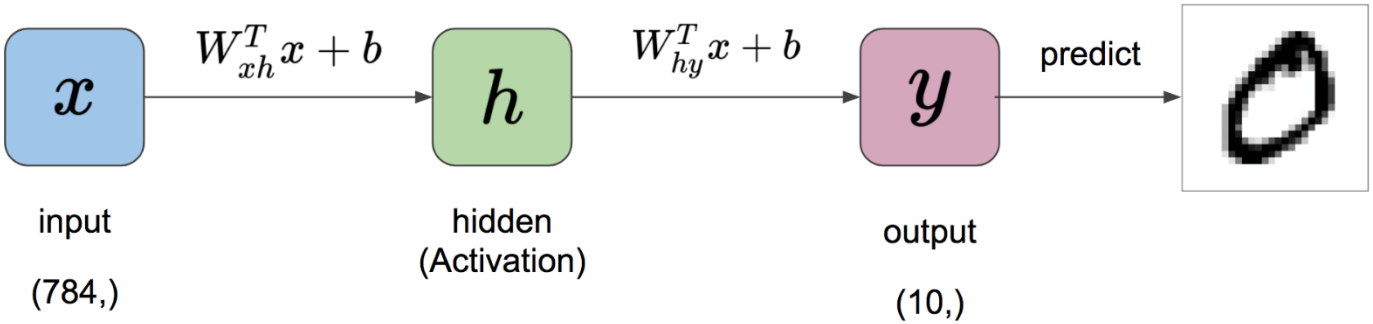
<사진설명 붙이기>
사진 → 단어들

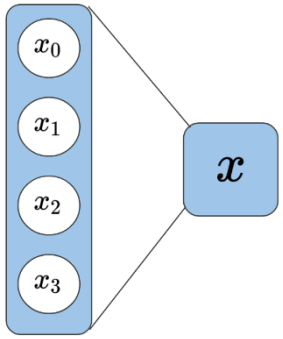
<감성분석>
단어들 → 감성점수

<번역>
단어들 → 단어들

기본 신경망 구조

- **MNIST 손글씨**



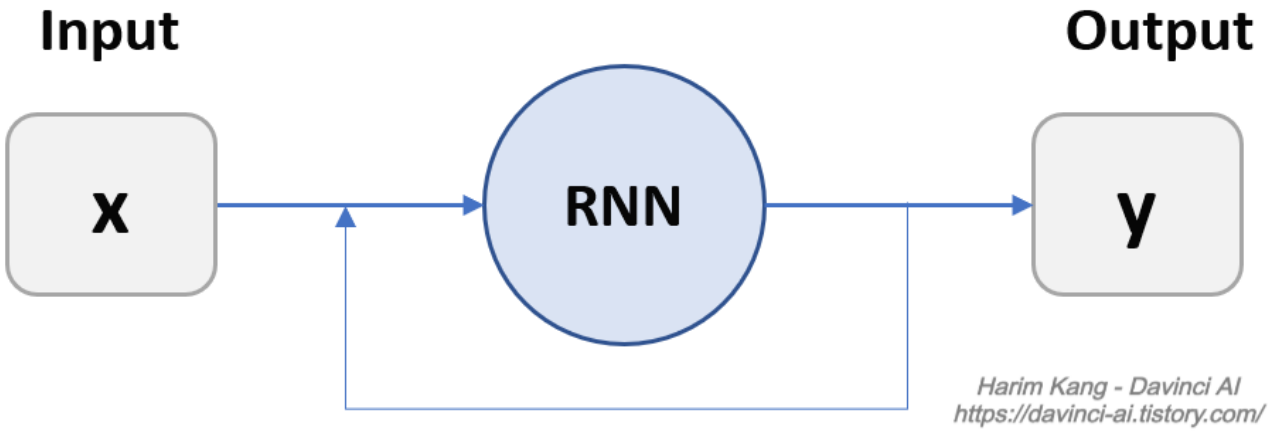


– 첫 번째 데이터(x1)와 그 다음 데이터(x2 등) 간의 구조는 독립적
  - **input x가 선형 결합 후, Hidden에 Activation function을 거쳐 다시 선형 결합을 통해 Output y를 구해 예측하는 알고리즘**

https://simonjisu.github.io/numpyseries/2018/03/07/rnnlstm.html

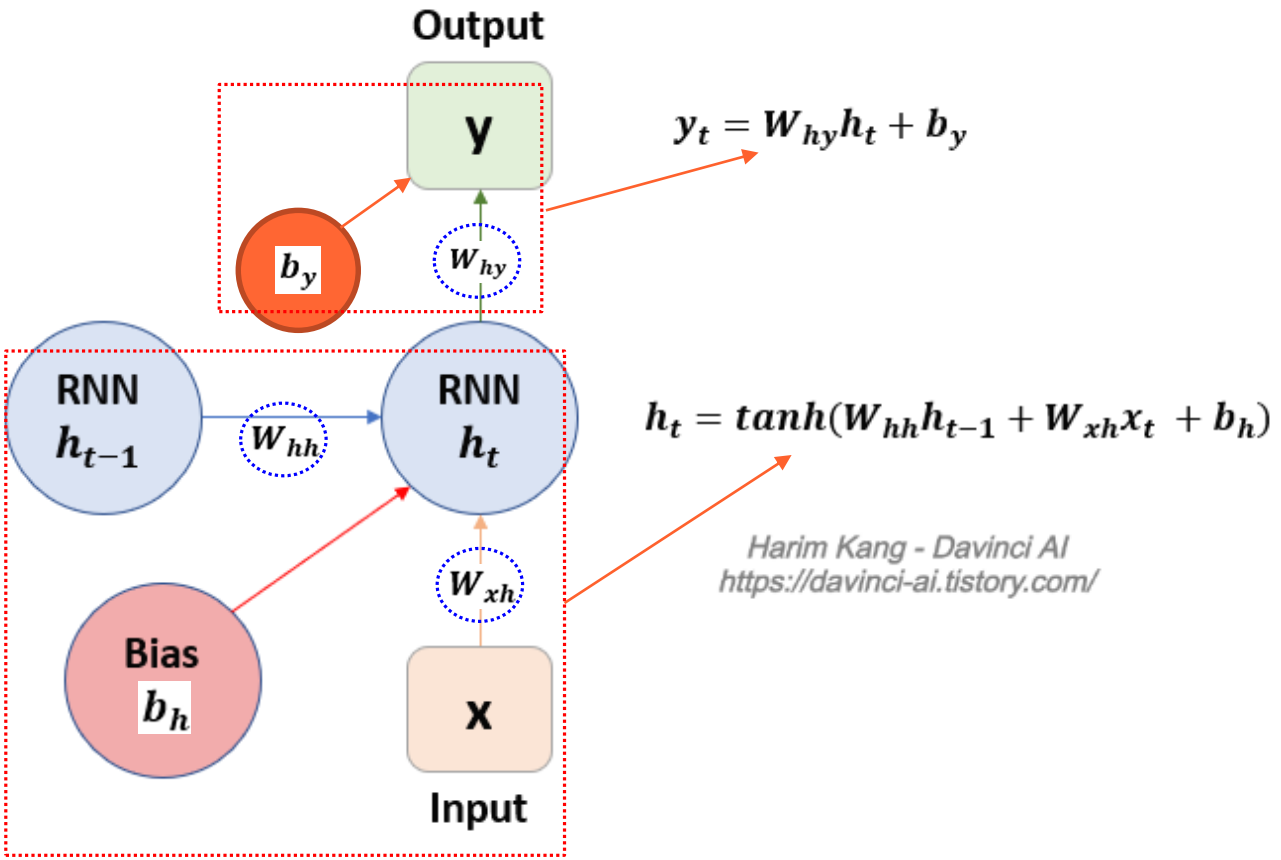
RNN 구조와 특징

- **RNN은 순환 구조**
  - 레이어의 출력을 다시 입력으로 받아서 사용
    - **이전의 데이터가 함께 결과에 영향을 미침**
- **RNN은 입력과 출력의 길이에 제한이 없음**
  - 구조를 바꾸면 다양한 형태, 스타일의 네트워크를 형성
  - 기본적으로 Fully connected 구조



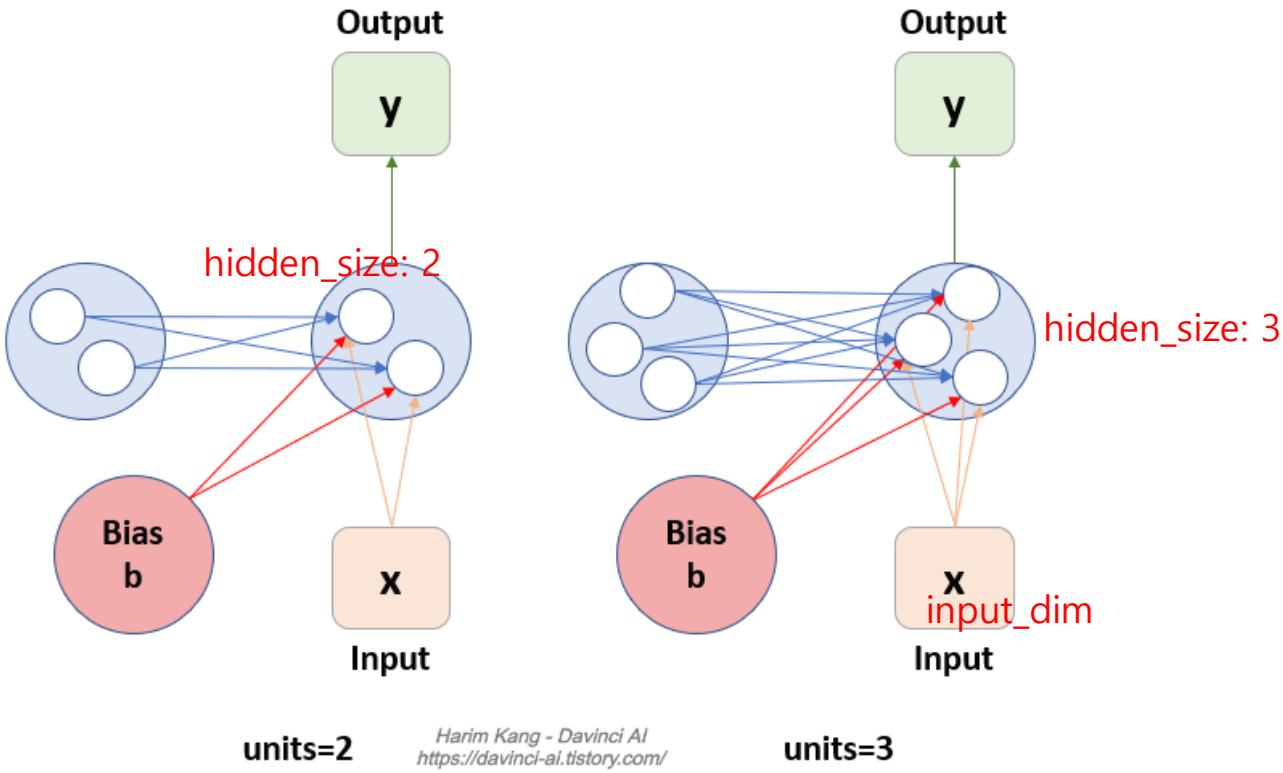
RNN의 가중치와 편향

- **가중치 부류**
  - 총 3개 부류
    - **Wxh**
    - **Whh**
    - **Why**
- **편향**
  - 총 2개 부류
    - **bh**
    - **by**



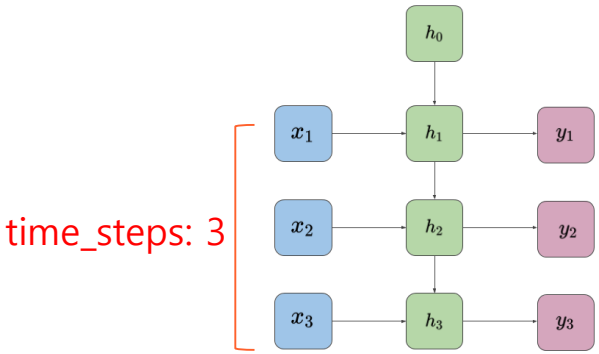
RNN 파라미터

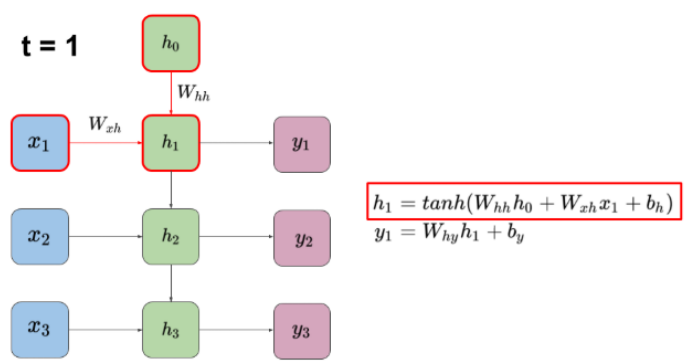
- **RNN은 모든 시간대에 동일한 변수를 사용**
  - units 패러미터는 RNN 신경망에 존재하는 뉴런의 개수
    - **SimpleRNN(hidden_size, input_shape=(timesteps, input_dim)))**
    - **SimpleRNN(units=10, input_shape=[4,1])**



Time step(t) 3인 RNN

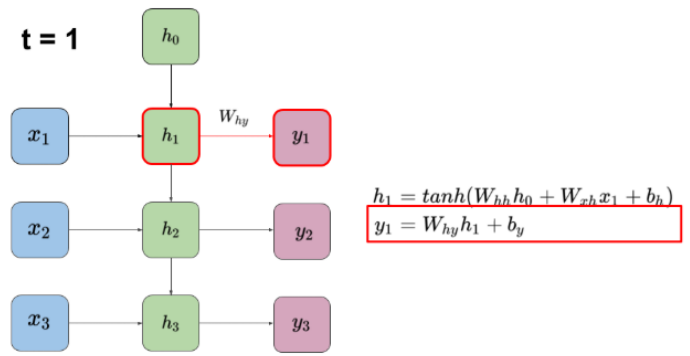
- **Time step = 0**
  - 각각 Layer들의 Weight를 초기화
    - **h0 층은 0으로, 나머지는 Xavier 가중치 초기값으로 초기화**
  - 또한 각 가중치는 각각 layer에서 공유





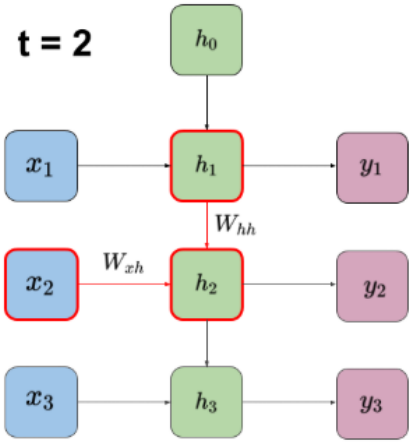
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

$for\ t\ in\ T$



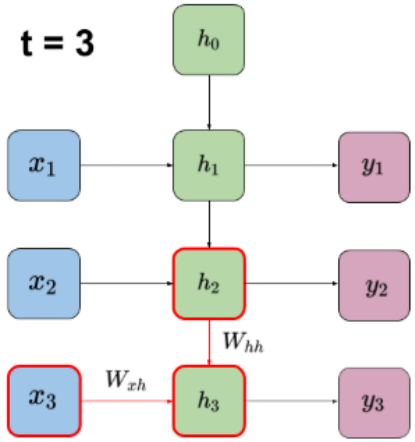
Time step(T) 2, 3, 단계

- t = 2



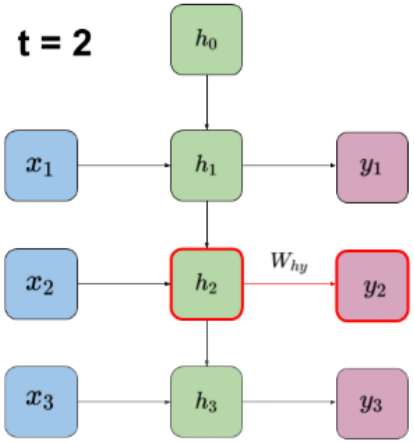
$$h_2 = tanh(W_{hh}h_1 + W_{xh}x_2 + b_h)$$
$$y_2 = W_{hy}h_2 + b_y$$

- t = 3



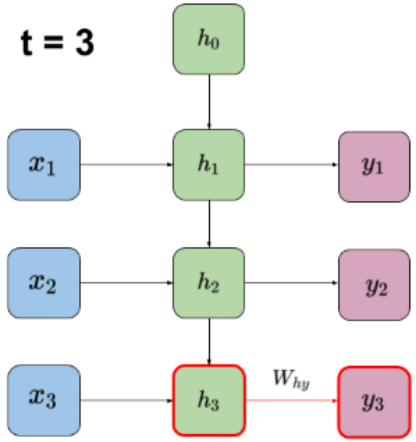
$$h_3 = tanh(W_{hh}h_2 + W_{xh}x_3 + b_h)$$
$$y_3 = W_{hy}h_3 + b_y$$

- t = 2



$$h_2 = tanh(W_{hh}h_1 + W_{xh}x_2 + b_h)$$
$$y_2 = W_{hy}h_2 + b_y$$

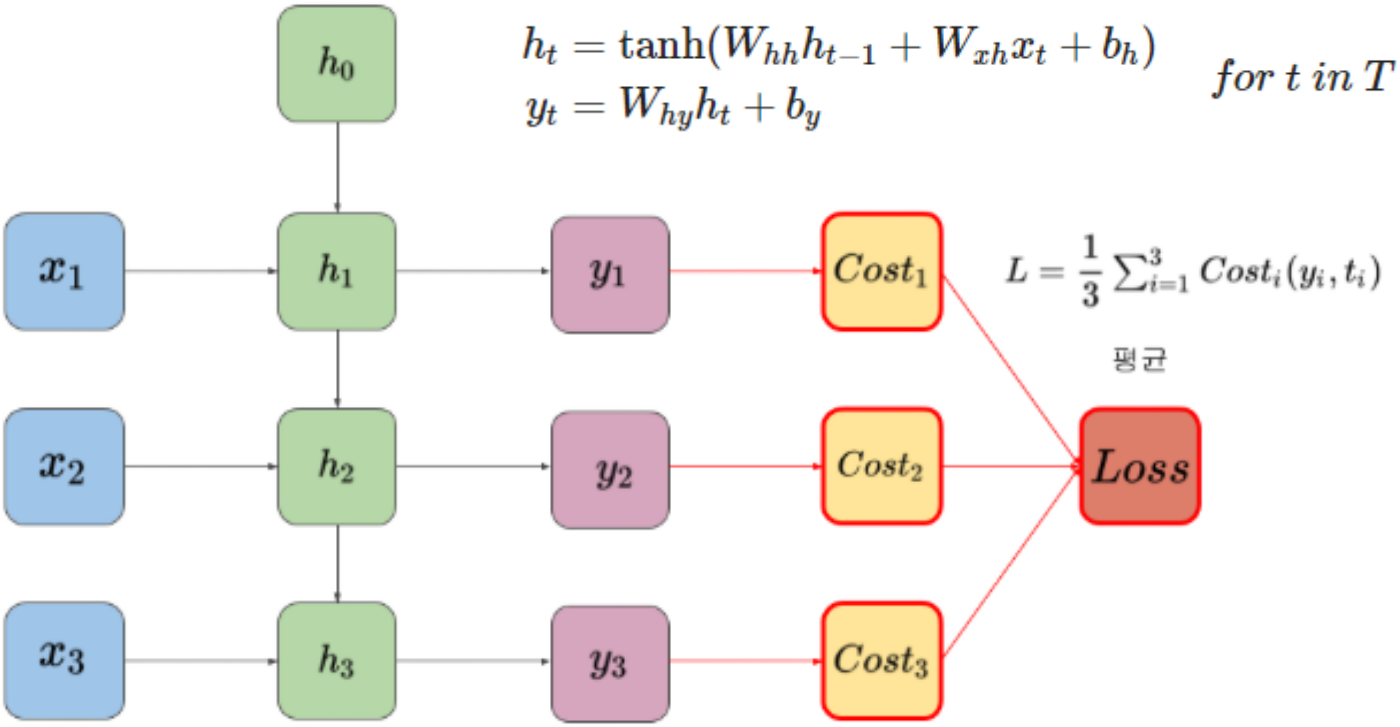
- t = 3



$$h_3 = tanh(W_{hh}h_2 + W_{xh}x_3 + b_h)$$
$$y_3 = W_{hy}h_3 + b_y$$

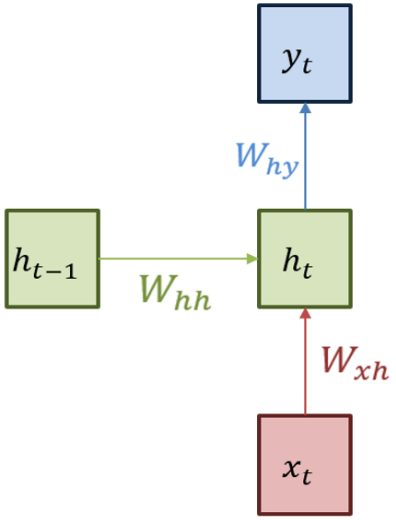
손실 함수

- **최종 Cost**
  - 모든 Cost Function의 평균



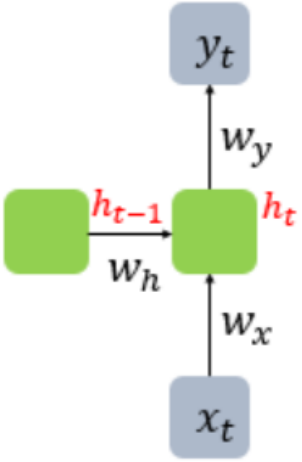
RNN의 연산

- **현재 시점 t에서의 은닉 상태값을 ht**
  - 은닉층: 메모리 셀 ht를 계산
    - **총 두 부분의 가중치**
      - **하나는 입력층에서 입력값을 위한 가중치 Wxh**
      - **하나는 이전 시점 t-1의 은닉 상태 값인 ht−1을 위한 가중치 Whh**
  - 출력층: yt 계산
    - **한 부분의 가중치 Why**



$$y_t = W_{hy}h_t + b_y$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$



은닉층 : $h_t = tanh(W_x x_t + W_h h_{t-1} + b)$

출력층 : $y_t = f(W_y h_t + b)$

단, f는 비선형 활성화 함수 중 하나.

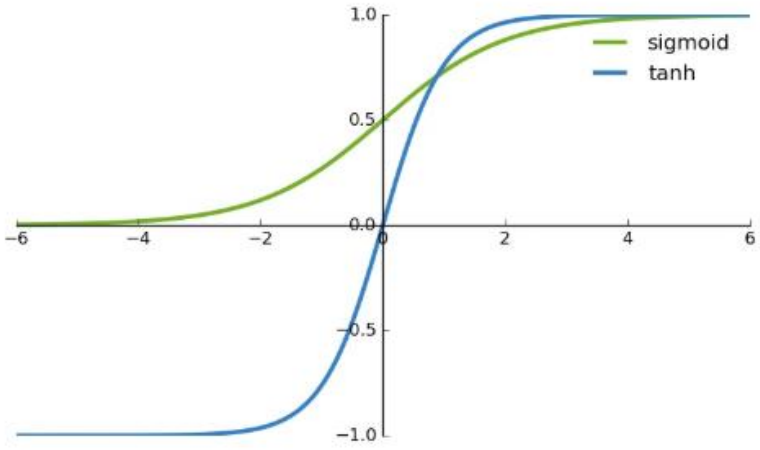
7장 순환 신경망

1 SimpleRNN layers

SimpleRNN 레이어

- **간단한 형태의 RNN 레이어**
  - 활성화 함수
    - **tanh 사용**
      - **실수 입력을 받아 -1에서 1사이의 출력 값을 반환**
      - **ReLU 같은 다른 활성화 함수도 사용**

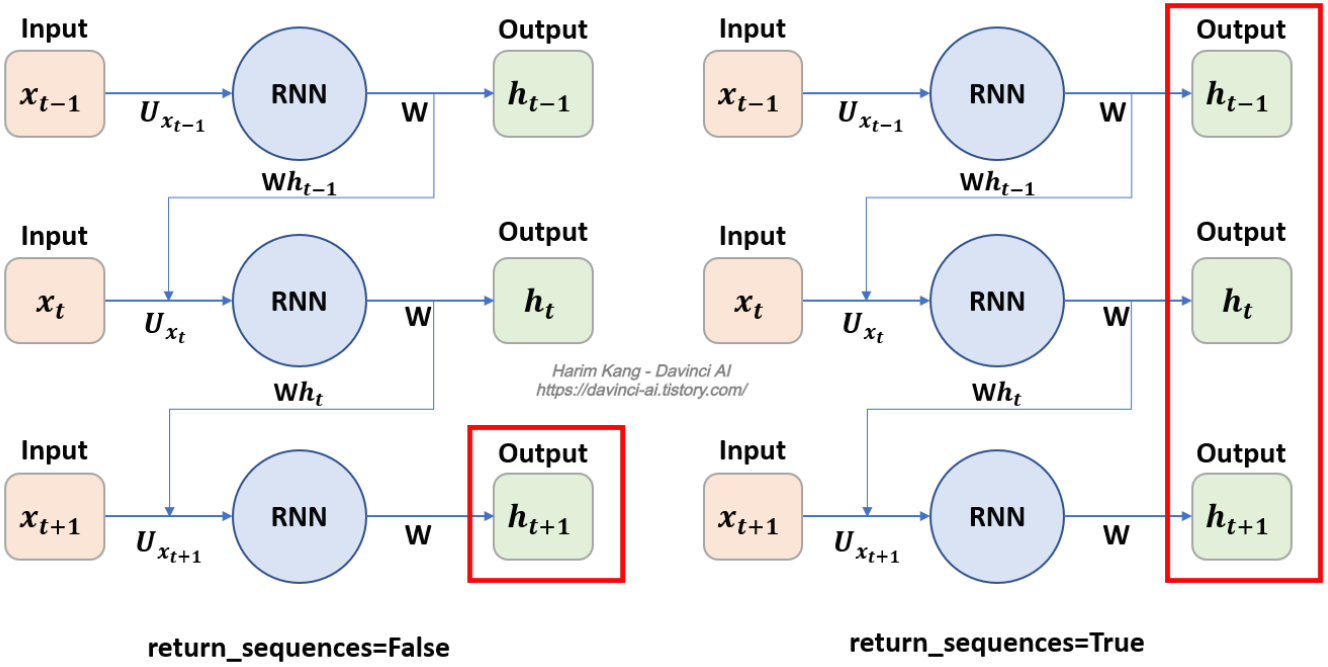
$$a = tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



- **코드**
  - rnn1 = tf.keras.layers.SimpleRNN(units=1, activation='tanh', return_sequences=True)
    - **units**
      - **SimpleRNN의 레이어에 존재하는 뉴런의 수**
    - **return_sequences**
      - **출력으로 시퀀스 전체를 출력할지 여부를 나타내는 옵션**
      - **여러 개의 RNN 레이어를 쌓을 때 사용**

인자 return_sequences

- **return_sequences**
  - RNN 계산 과정에 있는 hidden state를 출력할 것인지에 대한 값을 의미
  - return_sequences = True
    - **다층으로 이루어진 RNN**
    - **one-to-many, many-to-many 출력을 위해서 사용**



파일

- **ch7_RNN_study.ipynb**

시퀀스 예측 모델 예제

- **4개의 숫자로 그 다음에 숫자를 예측**
  - [0.0 0.1 0.2 0.3]이라는 0.1씩 늘어나는 수열을 줄 때
    - **input_shape=[4, 1]**
      - **step_size 4**
      - **한 순간의 자료 0.0 인 input_dim 벡터 1**
  - 이후의 값 0.4를 예측하는 네트워크
    - **출력인 결과 벡터 1**
- **units**
  - hidden_size 10은 하이퍼패러미터
- **return_sequences=False**
  - 중간 결과는 사용 안함

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=10, return_sequences=False, input_shape=[4,1]),
    tf.keras.layers.Dense(1)
])
```

학습과 테스트 데이터 생성

```
X = []
Y = []

for i in range(6):
  # [0, 1, 2, 3], [1, 2, 3, 4]
  lst = list(range(i,i+4))

  # 위에서 구한 시퀀스의 숫자들을 각각 10으로 나눈 다음 저장합니다.
  # SimpleRNN에 각 타임스텝에 하나씩 숫자가 들어가기 때문에
    여기서도 하나씩 분리해서 배열에 저장합니다.
  X.append(list(map(lambda c:[c/10], lst)))

  # 정답에 해당하는 4, 5 등의 정수 역시 앞에서처럼 10으로
    나눠서 저장합니다.
  Y.append((i+4)/10)

X = np.array(X)
Y = np.array(Y)

for i in range(len(X)):
  print(X[i], Y[i])
```

```
[[0. ]
 [0.1]
 [0.2]
 [0.3]] 0.4
[[0.1]
 [0.2]
 [0.3]
 [0.4]] 0.5
[[0.2]
 [0.3]
 [0.4]
 [0.5]] 0.6
[[0.3]
 [0.4]
 [0.5]
 [0.6]] 0.7
[[0.4]
 [0.5]
 [0.6]
 [0.7]] 0.8
[[0.5]
 [0.6]
 [0.7]
 [0.8]] 0.9
```

모델 정의

- **SimpleRNN 레이어를 사용한 네트워크를 정의**
  - units = 10
    - **RNN 신경망의 은닉 층에 존재하는 뉴런의 개수**
  - input_shape = [4,1]
    - **입력 형태 = [time_step, input_dim]**
  - return_sequences=False
    - **RNN 계산 과정에 있는 hidden state를 출력 사용하지 않음**
- **출력을 위한 시퀀셜 모델**
  - Dense 레이어가 뒤에 추가
    - **활성화 함수 없이 사용**

```
[[0. ]
 [0.1]
 [0.2]
 [0.3]]
```

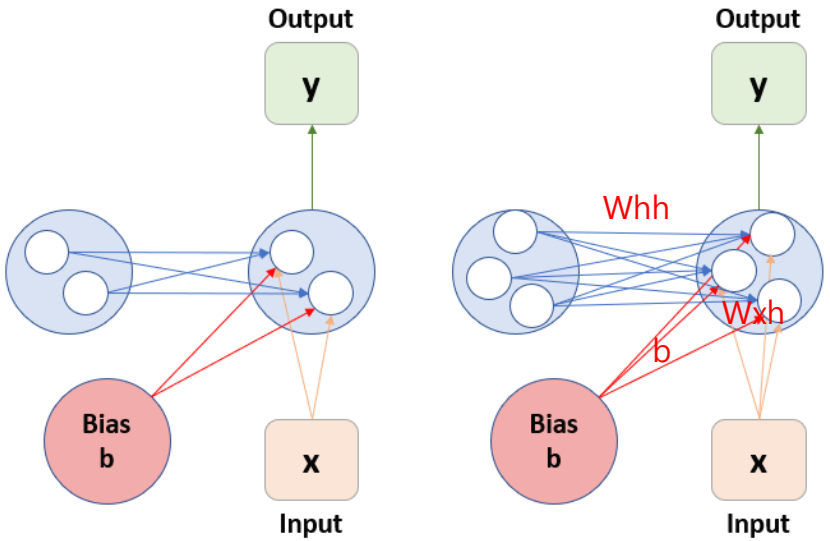
```
# 7.3 시퀀스 예측 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=10, return_sequences=False, input_shape=[4,1]),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

모델 RNN의 뉴런 수 units

- 중간층의 뉴런 수가 10개
  - 입력
    - Whh 10개, 편향 b 1개, input 1개
  - 출력
    - 중간층 뉴런 수 10개

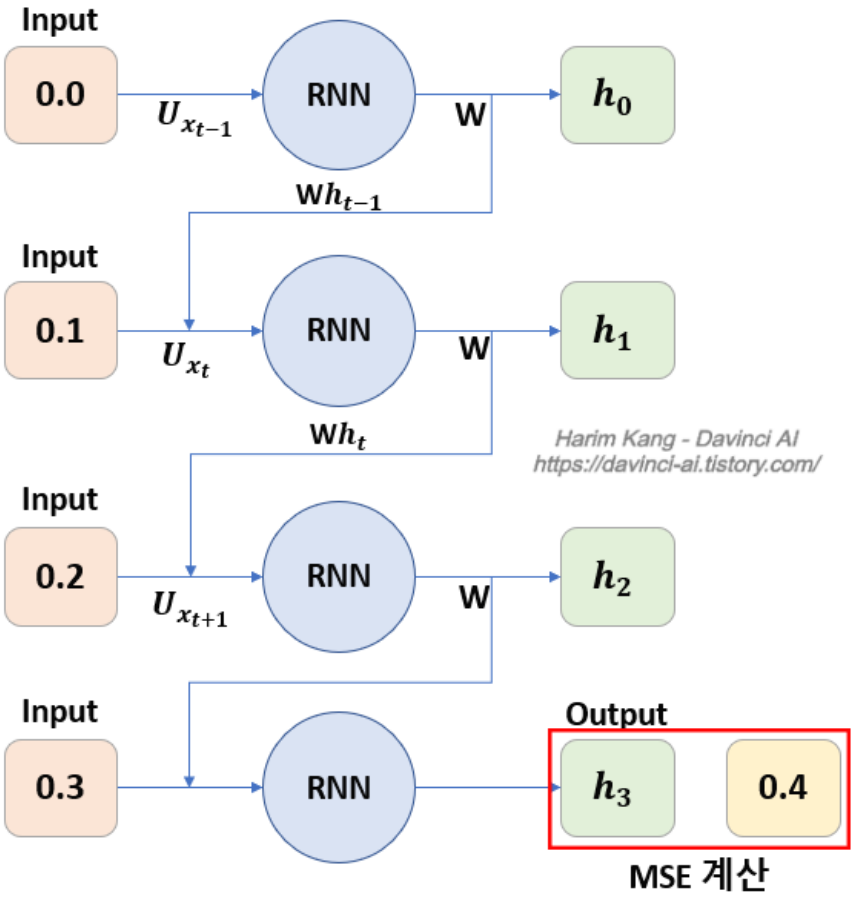
```
# 7.3 시퀀스 예측 모델 정의
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=10, return sequences=False, input shape=[4,1]),
    tf.keras.layers.Dense(1)
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 10)                120   ?
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================
Total params: 131
Trainable params: 131
Non-trainable params: 0
_________________________________________________________________
```



모델 RNN의 입력 형태 input_shape

- **input_shape=[4, 1]**
  - [4,1]은 각각 timesteps, input_dim
    - **[[0.]**
    - **[0.1]**
    - **[0.2]**
    - **[0.3]]**
  - 타입스텝(timesteps)
    - **순환 신경망이 입력에 대해 계산을 반복하는 횟수**
  - input_dim
    - **입력 벡터의 크기, 1**
  - 시퀀스 예측 모델은 4 타임 스텝에 걸쳐 입력을 받고,
    - **마지막에 출력 값을 다음 레이어로 반환**
- **Dense 레이어**
  - 별도의 활성화함수가 없음
  - 손실 함수
    - **이 값과 0.4와의 차이가 mse**
      - **평균 제곱 오차(Mean Squared Error)**



학습과 예측

- **6개 문제에 대한 예측 값**
  - 정답은 4, 5, 6, 7, 8, 9

```
model.fit(X, Y, epochs=100, verbose=0)
print(model.predict(X))
```

```
[[0.37758377]
 [0.5054759 ]
 [0.61844903]
 [0.71716857]
 [0.8032897 ]
 [0.8785695 ]]
```

```
1 print(model.predict(np.array([[[0.6], [0.7], [0.8], [0.9]]])))
2 print(model.predict(np.array([[[-0.1], [0.0], [0.1], [0.2]]])))
```

```
[[0.94448555]]
[[0.23629726]]
```

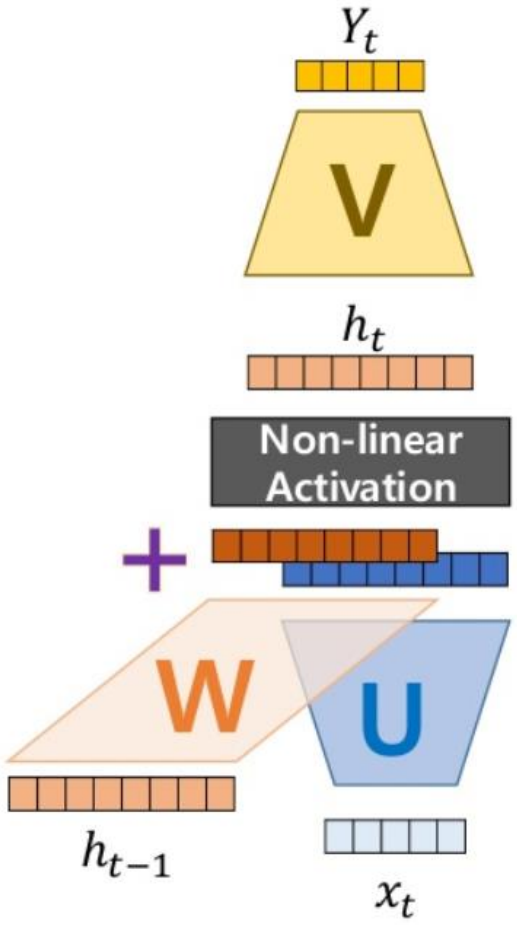
순환신경망 RNN

7장 순환신경망

패러미터 수 계산

RNN 패러미터 계산 개요

- 전통적 RNN(vanilla RNN)



Vanilla RNN

$$h_t = f(Ux_t + Wh_{t-1})$$

tanh 현재입력 과거기억

$$Y_t = g(Vh_t)$$

$g(x) = x$

RNN의 패러미터 수

- **simpleRNN 층**
  - (units + input_dim + 1(bias)) x units
    - **중간 층 뉴런 수(units=2), 입력 차원(input_dim) 1, : (2+1+1) * 2 = 8**
- **Dense 층**
  - (units + 1(bias)) x Dense층 출력 수
    - **(2 + 1) * 1 = 3**

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=2,
                              input_shape=[4,1],
                            return_sequences=False),
    tf.keras.layers.Dense(1)
])
```

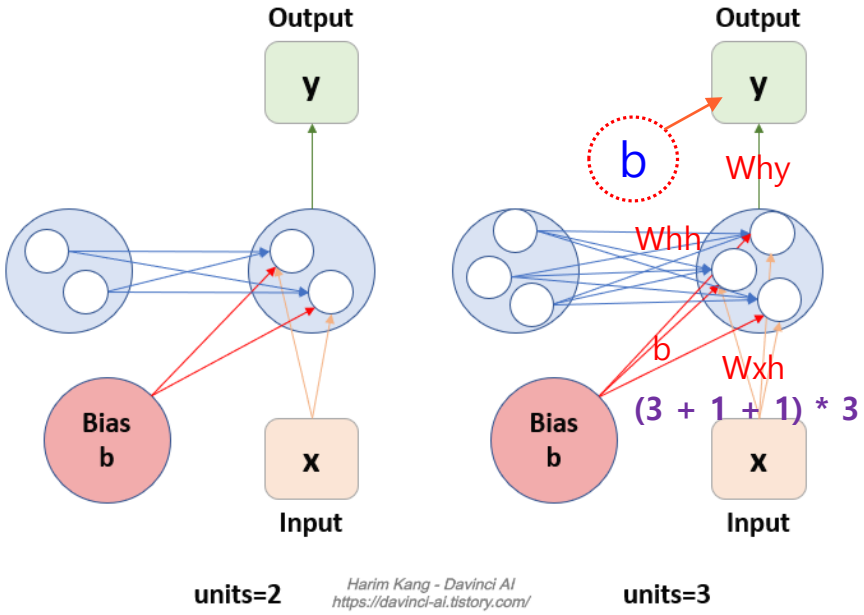
```
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_12 (SimpleRNN)    (None, 2)                 8
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 3
=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
```

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=3,
                              input_shape=[4,1],
                            return_sequences=False),
    tf.keras.layers.Dense(1)
])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_14 (SimpleRNN)    (None, 4, 3)              15
_________________________________________________________________
dense_14 (Dense)             (None, 4, 1)              4
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
```

(3 + 1 + 1) * 3

(3 + 1) * 1

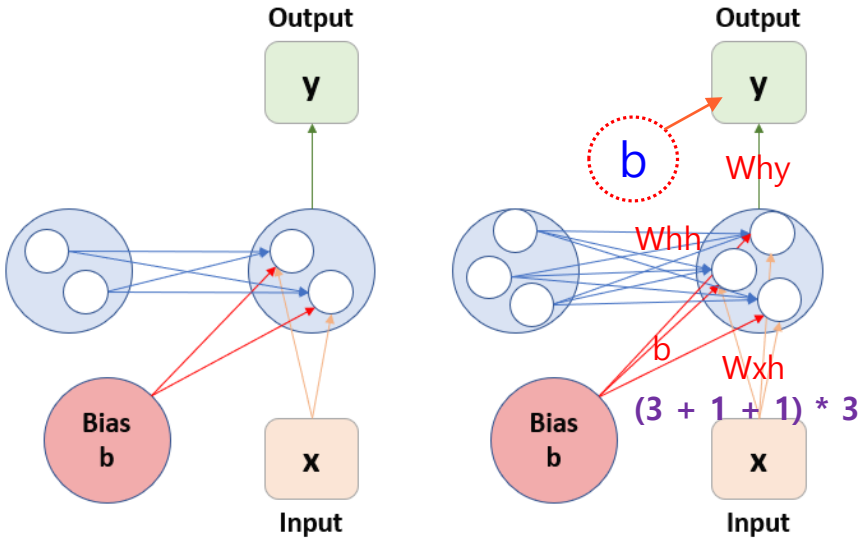


RNN의 패러미터 수(2)

- **simpleRNN 층**
  - (units + input_dim + 1(bias)) x units
    - **증간 층 뉴런 수(units=10), 입력 차원(input_dim) 3, : (10+1+3) * 10 = 140**
- **Dense 층**
  - (units + 1(bias)) x Dense층 출력 수
    - **(10 + 1) * 2 = 22**

```
model = tf.keras.Sequential([
    tf.keras.layers.SimpleRNN(units=10,
                      input_shape=[4,3],
                return_sequences=False),
    tf.keras.layers.Dense(2)
])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_15 (SimpleRNN)    (None, 10)                140
_________________________________________________________________
dense_15 (Dense)             (None, 2)                 22
=================================================================
Total params: 162
Trainable params: 162
Non-trainable params: 0
```

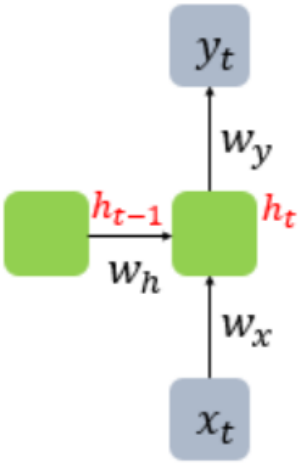


units=2 units=3

RNN의 벡터와 행렬의 크기

- **입력 Xt**
  - d: 입력 벡터의 차원
    - **예, [[1], [2], [3], [4]] : 1, [[1, 2], [2, 3], [3, 4], [4, 5]] : 2**
- **Dh**
  - 은닉 상태의 크기, units 수
- **각각의 가중치 Wx, Wh, Wy**
  - 모든 시점에서 값을 동일하게 공유
    - **은닉층이 2개 이상일 경우에는 은닉층 2개의 가중치는 서로 다름**

$x_t : (d \times 1)$
$W_x : (D_h \times d)$
$W_h : (D_h \times D_h)$
$h_{t-1} : (D_h \times 1)$
$b : (D_h \times 1)$



은닉층 : $h_t = tanh(W_x x_t + W_h h_{t-1} + b)$
출력층 : $y_t = f(W_y h_t + b)$
단, f는 비선형 활성화 함수 중 하나.

$$\tanh\left(W_h \times h_{t-1} + W_x \times x_t + b \right) = h_t$$

W_h : $D_h \times D_h$, h_{t-1} : $D_h \times 1$, W_x : $D_h \times d$, x_t : $d \times 1$, b : $D_h \times 1$, h_t : $D_h \times 1$

https://wikidocs.net/22886