Recurrent Neural Network (RNN)

집현전 2기 초급 11조
이기동 이주형 최선웅

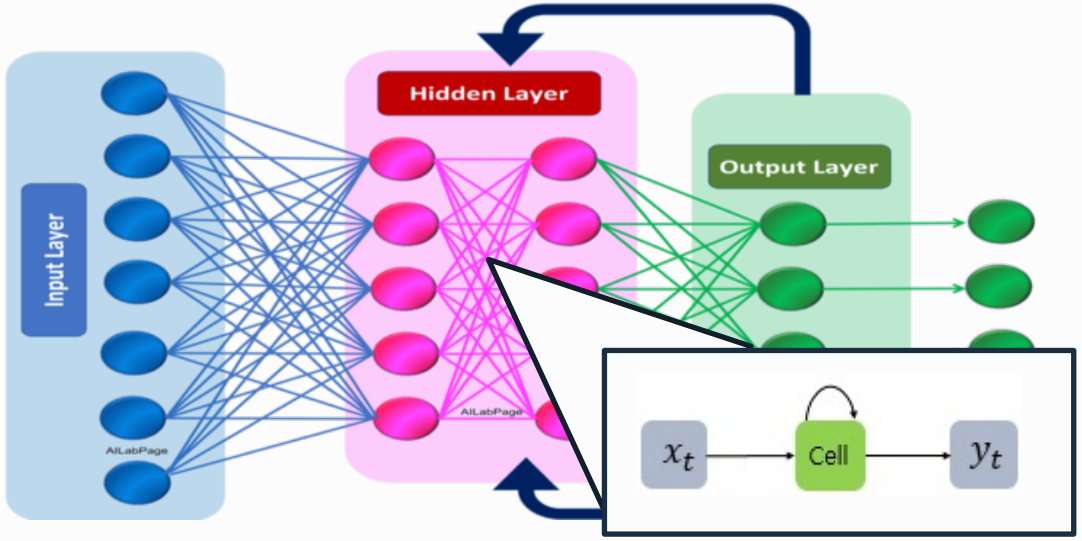
목차

Chapter 1. RNN

RNN의 개요
활용방안 및 응용 모델

1. RNN (Recurrent Neural Network)

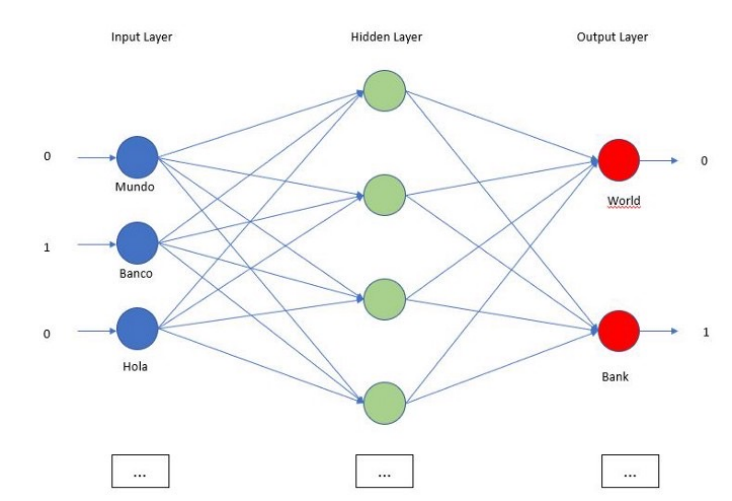
RNN의 정의



- 은닉층의 활성함수 결과값을 재귀 적용하여 피드백
- 입력층(Input Layer), 은닉층(Hidden Layer), 출력층(Output Layer)
- 메모리 셀(RNN 셀): 재귀 적용되는 결과값을 저장하는 노드

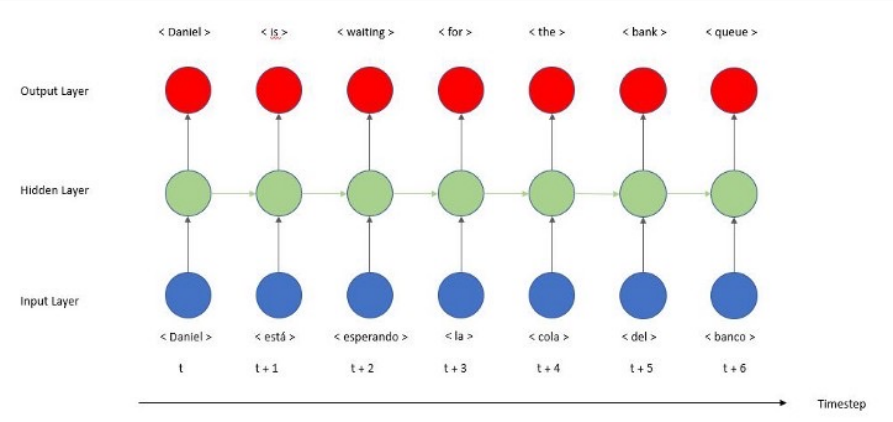
2. RNN과 기존 FFNN과의 비교

- **피드 포워드 신경망(FFNN)**



- 시간당 입력값에 대한 연산이 해당 시간의 출력값에만 영향(단방향성)
- 처리속도는 빠르나, 시간별 연관성을 배제하여 이전 상태의 결과가 영향을 주지 못함

- **순환 신경망(RNN)**



- 은닉층의 결과값을 메모리셀을 이용해 저장하여 다음 상태에서의 연산에 활용
- 이전 상태의 결과값을 반영하여 분석시 입력값에 대한 복합적 고려 가능

2. RNN과 기존 FFNN과의 비교

- **피드 포워드 신경망(FFNN)**

1) 입력값에 대한 모델기반 단방향 분석에 용이

2) 처리속도가 빠르고 단순함

3) 입력정보간의 연관성을 고려하지 않음

독립된 입력정보에 대한
병렬 연산에 우수

(개별 사진에 대한 강아지or고양이 판단)

- **순환 신경망(RNN)**

1) 입력값에 대한 개별 분석값을 타 입력에 대한 분석시 재활용

2) 처리구조가 복잡하고 상대적으로 느림

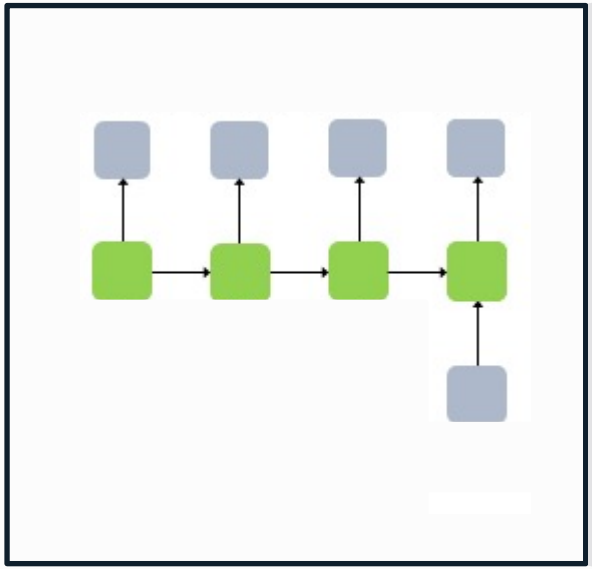
3) 입력정보간의 연관성 존재시 결과 정확도 우수

입력정보간에 연관성 존재시 이에 대한
다중 분석에 우수

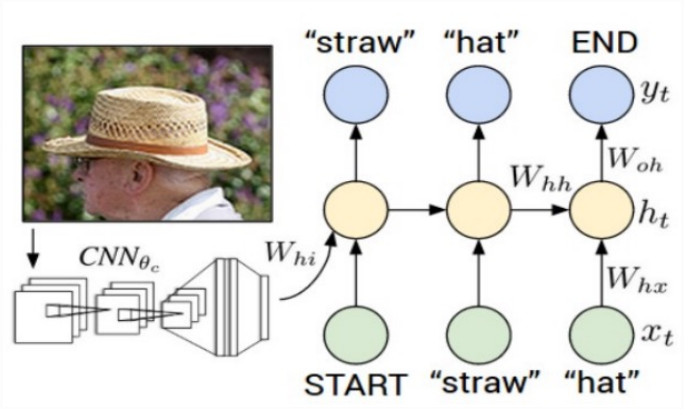
(문장에 대한 품사 태깅, 감성 분석)

3. RNN의 활용 – 입출력별 분류(1)

- 단일입력 대비 다중 출력(One-to-Many) 모델



이미지 캡셔닝



- 이미지 입력에 대한 사진제목 출력
- 추출된 단어에 대해 중요도별로 나열

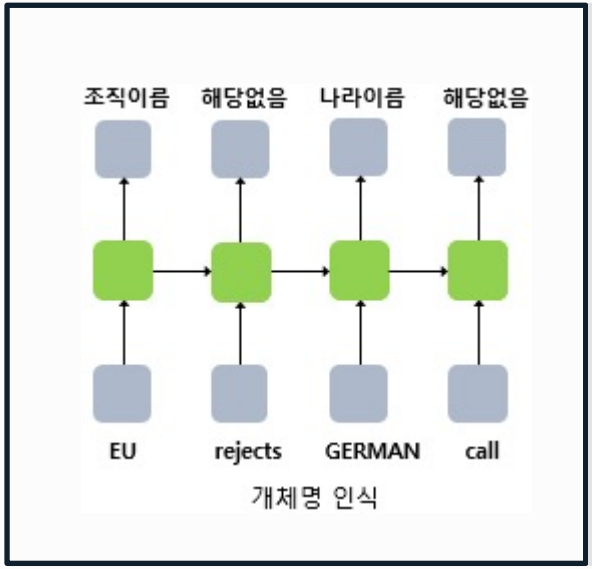
3. RNN의 활용 – 입출력별 분류(2)

스팸메일 필터링

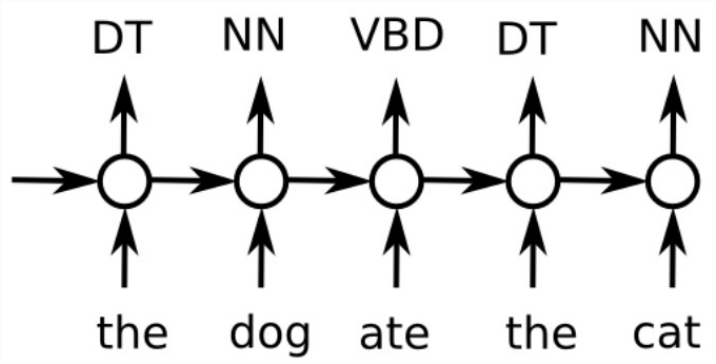
텍스트(메일의 내용)	레이블(스팸 여부)
당신에게 드리는 마지막 혜택! ...	스팸 메일
내일 뵐 수 있을지 확인 부탁...	정상 메일
쉿! 혼자 보세요...	스팸 메일
언제까지 답장 가능할...	정상 메일
...	...
(광고) 멋있어질 수 있는...	스팸 메일

- 여러 단어를 입력받아 연관도를 판단
- 최종 결과로 0/1 판단(스팸/정상) 출력

3. RNN의 활용 – 입출력별 분류(3)

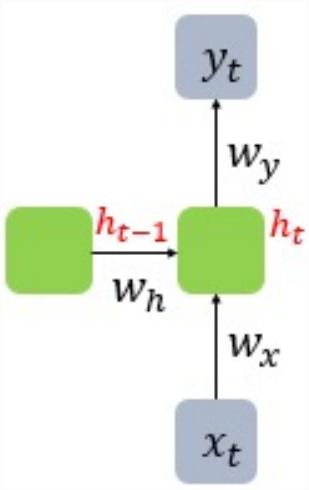


품사 태깅



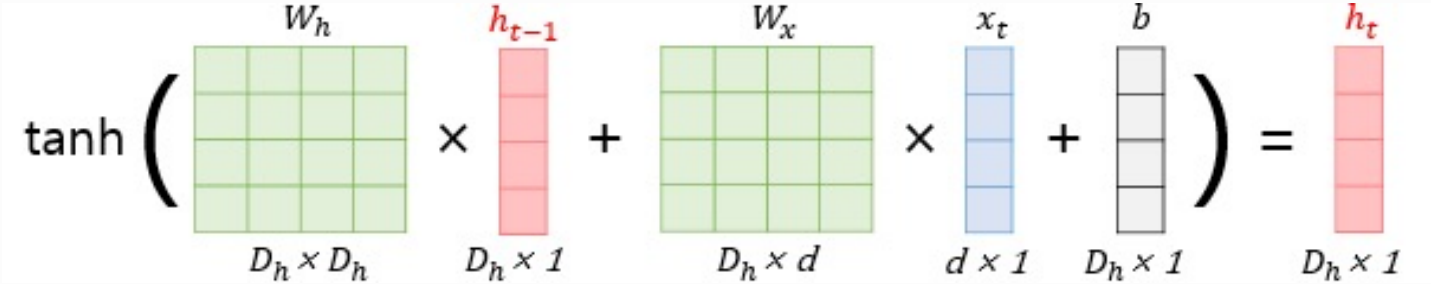
- 문장 입력에 대해 단어간 연관성 분석
- 각 단어 입력에 대한 품사 태그 출력

4. RNN 계산 수식



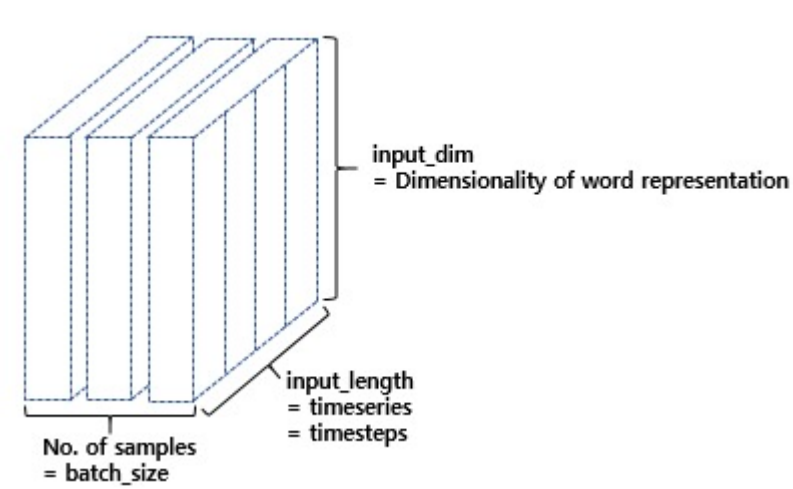
- 은닉층 : $h_t = \tanh(W_y x_t + W_h h_{t-1} + b)$
- 출력층 : $y_t = f(W_y h_t + b)$
  (단, f는 비선형 활성화 함수)

- xt : (d×1)
- Wx : (Dh×d
- Wh : (Dh×D
  h
- ht−1 : (Dh×
  1)
- b : (Dh×1)



5. RNN의 활용 - Keras

- **Keras 적용 코드**




```
# RNN 층을 추가하는 코드.
model.add(SimpleRNN(hidden_size)) # 가장 간단한 형태
```

```
# 추가 인자를 사용할 때 model.add(SimpleRNN(hidden_size,
input_shape=(timesteps, input_dim))) # 다른 표기
model.add(SimpleRNN(hidden_size, input_length=M,
input_dim=N)) # 단, M과 N은 정수
```

* hidden_size = 은닉 상태의 크기를 정의. 메모리 셀이 보내는 값과 동일 (보통 128, 256, 512, 1024 등의 값을 가짐)
* timesteps = 입력 시퀀스의 길이(input_length)
* input_dim = 입력의 크기

- 최종시점의 은닉 상태(2D): batch_size, output_dim
- 전체 은닉상태 시퀀스(3D): batch_size, timesteps, output_dim

5. RNN의 활용 - Keras

- **2D 텐서 출력 모델**

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

```
model = Sequential() model.add(SimpleRNN(3,
input_shape=(2,10))) # model.add(SimpleRNN(3,
input_length=2, input_dim=10))와 동일함.
model.summary()
```

```
_________________________________________________
_____ Layer (type) Output Shape Param #
=================================
=============================
simple_rnn_1 (SimpleRNN) (None, 3) 42
=================================
=============================
Total params: 42 Trainable params: 42 Non-trainable
params: 0
_________________________________________________
_____
```

- **2D 텐서 – Batch_size 설정**

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

```
model = Sequential()
model.add(SimpleRNN(3,
batch_input_shape=(8,2,10)))
model.summary()
```

```
_________________________________________________
_____
Layer (type)                 Output Shape
Param #
=================================
=============================
simple_rnn_2 (SimpleRNN)     (8, 3)
42
=================================
=============================
Total params: 42
Trainable params: 42
Non-trainable params: 0
_________________________________________________
_____
```

- **3D 텐서 출력 모델**

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

```
model = Sequential()
model.add(SimpleRNN(3, batch_input_shape=(8,2,10),
return_sequences=True))
model.summary()
```

```
_________________________________________________
_____
Layer (type)                 Output Shape
Param #
=================================
=============================
simple_rnn_3 (SimpleRNN)     (8, 2, 3)                 42
=================================
=============================
Total params: 42
Trainable params: 42
Non-trainable params: 0
_________________________________________________
_____
```

5. RNN의 활용 - Keras

- **Pseudo code (RNN)**

```
hidden_state_t = 0 # 초기 은닉 상태를 0(벡터)로 초기화
for input_t in input_length: # 각 시점마다 입력을 받는다.
    output_t = tanh(input_t, hidden_state_t) # 각 시점에 대해서 입력과 은닉 상태를 가지고 연산
    hidden_state_t = output_t # 계산 결과는 현재 시점의 은닉 상태가 된다
```

- **Keras 적용 코드**

```
import numpy as np

timesteps = 10 # 시점의 수. NLP에서는 보통 문장의 길이가 된다.
input_dim = 4 # 입력의 차원. NLP에서는 보통 단어 벡터의 차원이 된다.
hidden_size = 8 # 은닉 상태의 크기. 메모리 셀의 용량이다.

inputs = np.random.random((timesteps, input_dim)) # 입력에 해당되는 2D 텐서

hidden_state_t = np.zeros((hidden_size,)) # 초기 은닉 상태는 0(벡터)로 초기화
# 은닉 상태의 크기 hidden_size로 은닉 상태를 만듬.
```

```
print(hidden_state_t) # 8의 크기를 가지는 은닉 상태. 현재는 초기 은닉 상태로 모든 차원이 0의 값을 가짐.
[0. 0. 0. 0. 0. 0. 0. 0.]
```

```
Wx = np.random.random((hidden_size, input_dim))  # (8, 4)크기의 2D 텐서 생성. 입력에 대한 가중치.
Wh = np.random.random((hidden_size, hidden_size)) # (8, 8)크기의 2D 텐서 생성. 은닉 상태에 대한 가중치.
b = np.random.random((hidden_size,)) # (8,)크기의 1D 텐서 생성. 이 값은 편향(bias).
```

```
print(np.shape(Wx))
print(np.shape(Wh))
print(np.shape(b))
```

```
(8, 4)
(8, 8)
(8,)
```

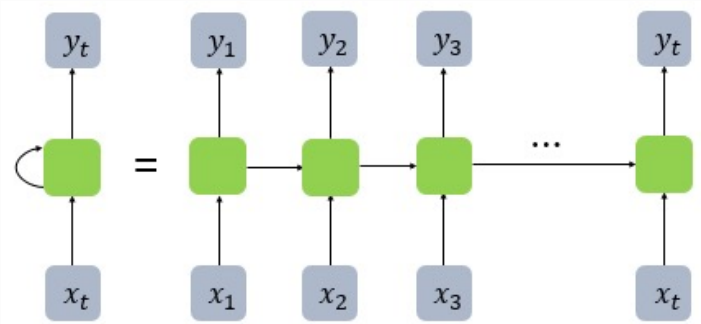
5. RNN의 활용 - Keras

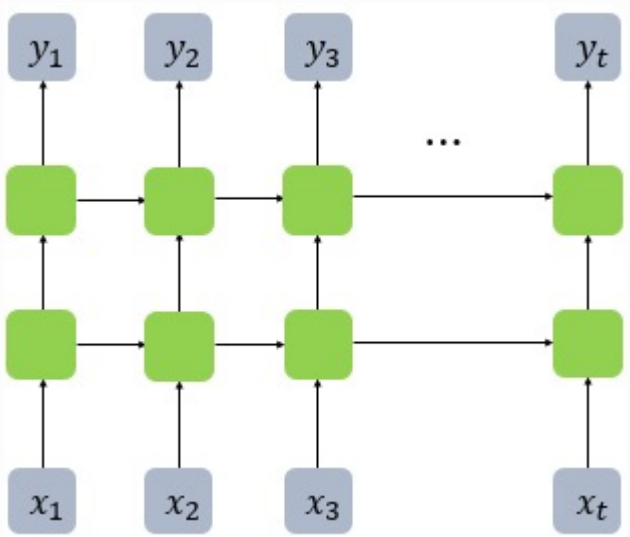
- **Keras 적용 코드 및 결과값**

```
total_hidden_states = []

# 메모리 셀 동작
for input_t in inputs: # 각 시점에 따라서 입력값이 입력됨.
  output_t = np.tanh(np.dot(Wx,input_t) + np.dot(Wh,hidden_state_t) + b) # Wx * Xt + Wh * Ht-1 +
b(bias)
  total_hidden_states.append(list(output_t)) # 각 시점의 은닉 상태의 값을 계속해서 축적
  print(np.shape(total_hidden_states)) # 각 시점 t별 메모리 셀의 출력의 크기는 (timestep, output_dim)
  hidden_state_t = output_t

total_hidden_states = np.stack(total_hidden_states, axis = 0)
# 출력 시 값을 깔끔하게 해준다.

print(total_hidden_states) # (timesteps, output_dim)의 크기. 이 경우 (10, 8)의 크기를 가지는 메모리 셀의
2D 텐서를 출력.
```

```
(1, 8)
(2, 8)
(3, 8)
(4, 8)
(5, 8)
(6, 8)
(7, 8)
(8, 8)
(9, 8)
(10, 8)
[[0.85575076 0.71627213 0.87703694 0.83938496
0.81045543 0.86482715 0.76387233 0.60007514]
 [0.99982366 0.99985897 0.99928638 0.99989791
0.99998252 0.99977656 0.99997677 0.9998397 ]
 [0.99997583 0.99996057 0.99972541 0.99997993
0.99998684 0.99954936 0.99997638 0.99993143]
 [0.99997782 0.99996494 0.99966651 0.99997989
0.99999115 0.99980087 0.99999107 0.9999622 ]
 [0.99997231 0.99996091 0.99976218 0.99998483
0.9999955  0.99989239 0.99999339 0.99997324]
 [0.99997082 0.99998754 0.99962158 0.99996278
0.99999331 0.99978731 0.99998831 0.99993414]
 [0.99997427 0.99998367 0.99978331 0.99998173
0.99999579 0.99983689 0.99999058 0.99995531]
 [0.99992591 0.99996115 0.99941212 0.99991593
0.999986   0.99966571 0.99995842 0.99987795]
 [0.99997139 0.99997192 0.99960794 0.99996751
0.99998795 0.9996674 0.99998177 0.99993016]
 [0.99997659 0.99998915 0.99985392 0.99998726
0.99999773 0.99988295 0.99999316 0.99996326]]
```

6. RNN의 확장 – 깊은(Deep) 순환신경망

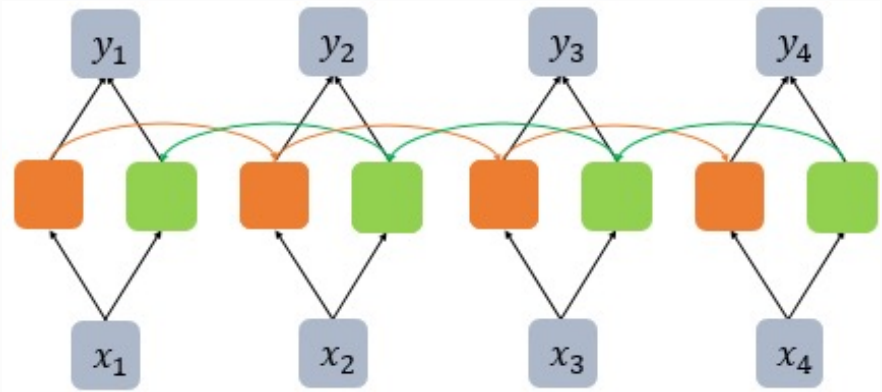





```
model = Sequential()
model.add(SimpleRNN(hidden_size, return_sequences = True))
model.add(SimpleRNN(hidden_size, return_sequences = True))
```

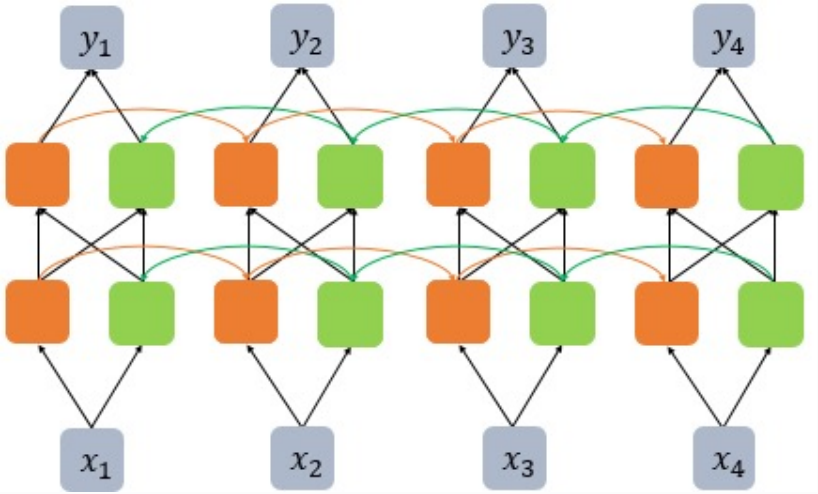
6. RNN의 확장 – 양방향 순환신경망

- **Single hidden Layer 기준**




```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Bidirectional

model = Sequential()
model.add(Bidirectional(SimpleRNN(hidden_size, return_sequences = True),
input_shape=(timesteps, input_dim)))
```

- **Mutliple hidden Layer 기준**



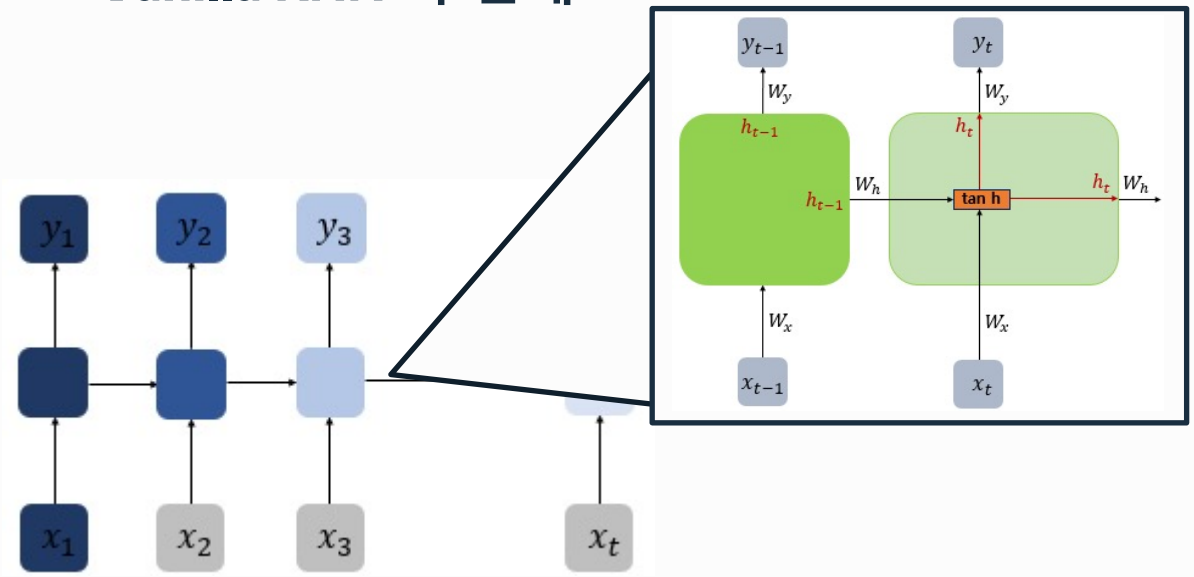

```
model = Sequential()
model.add(Bidirectional(SimpleRNN(hidden_size, return_sequences = True),
input_shape=(timesteps, input_dim)))
model.add(Bidirectional(SimpleRNN(hidden_size, return_sequences = True)))
model.add(Bidirectional(SimpleRNN(hidden_size, return_sequences = True)))
model.add(Bidirectional(SimpleRNN(hidden_size, return_sequences = True)))
```

Chapter 2. LSTM & GRU

LSTM 및 GRU의 정의 및 활용

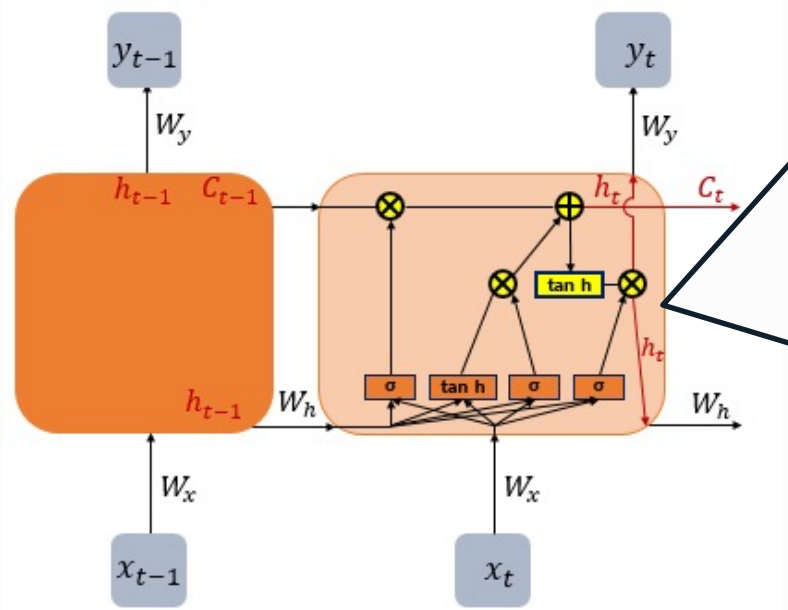
1. Vanilla RNN

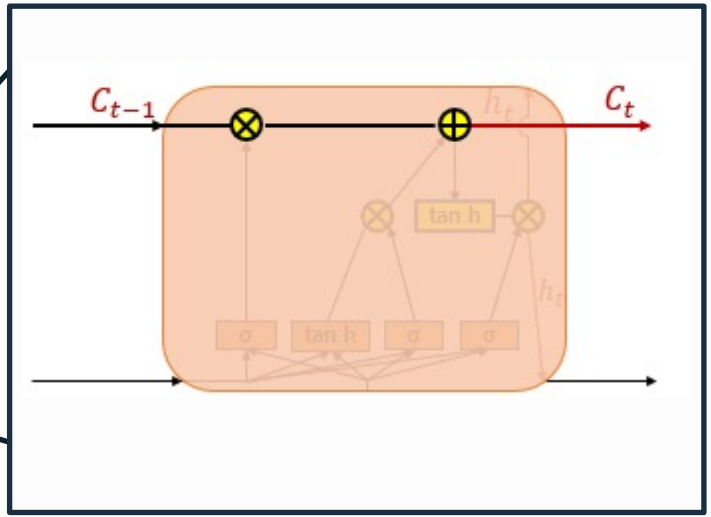
Vanilla RNN 의 한계



- 장기 의존성 문제
  - 짧은 Sequence에 대한 계산에서만 효과를 보임
  - RNN의 Time step이 길어질수록 앞의 정보의 영향력이 감소 (초기 정보에 대한 손실 발생)

2. LSTM(Long Short-Term Memory)



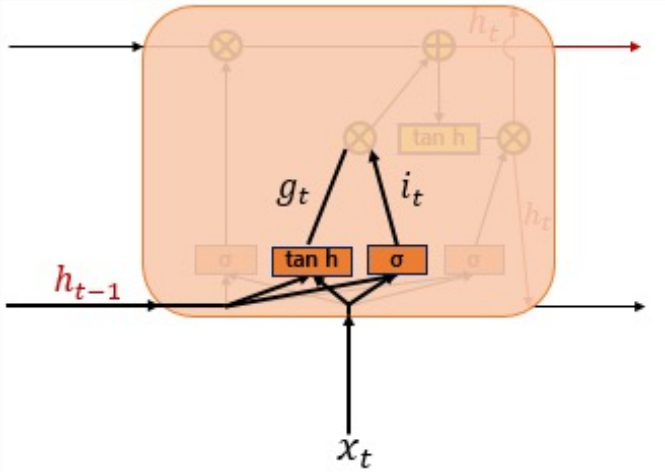


- Hidden layer 내 Cell에 Input/Forget Gate 추가
  - 연산 정보량을 조절하고 불필요한 정보 삭제

- 현재 시간(t) 기준 가중치(w)와 입력의 연산값에 이전(t-1) 출력값을 더하여 인풋 출력값 계산
  - t에서 받은 Input의 영향력을 매회 결정

2. LSTM(Long Short-Term Memory)

- **입력 게이트**

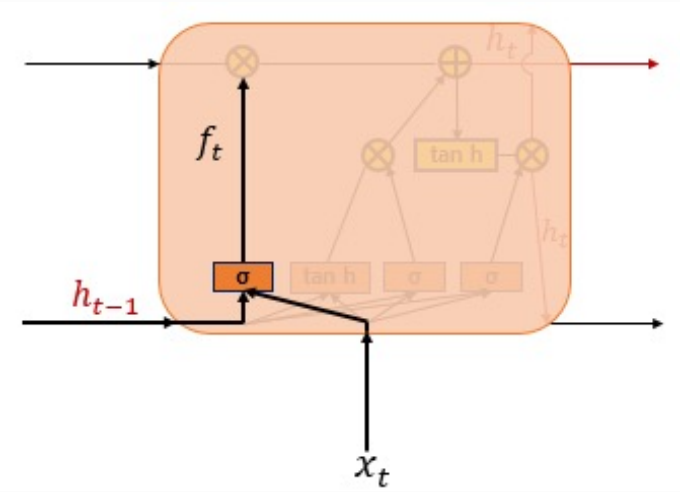


$i_{t}=\sigma(W_{xi}x_{t}+W_{hi}h_{t-1}+b_{i})$

$g_{t}=\tanh(W_{xg}x_{t}+W_{hg}h_{t-1}+b_{g})$

- 현재의 정보 기억. Wx 값을 시그모이드 함수를 지나 i값 계산
- t-1과의 가중치 연산값을 tanh함수를 씌워 나온 값을 통해 기억할 정보량을 설정

- **삭제 게이트**

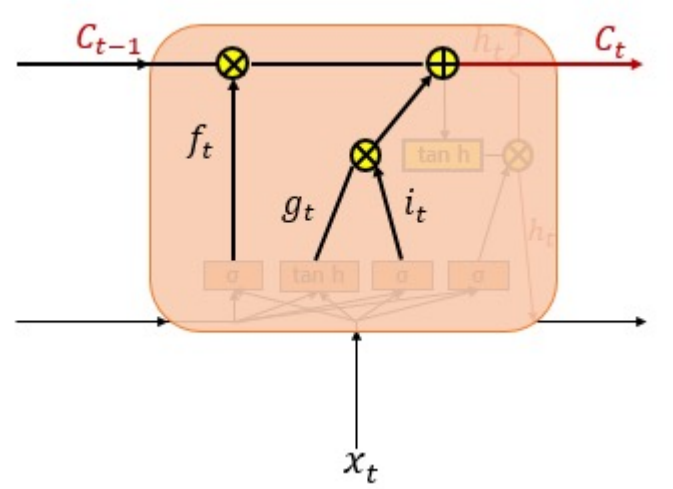


$f_{t}=\sigma(W_{xf}x_{t}+W_{hf}h_{t-1}+b_{f})$

- 기억을 삭제하기 위한 게이트로, f 값을 계산하여 은닉상태의 가중치 곱한 값으로 정보량의 삭제를 판단
- 불필요한 정보(0에 가까운 정보)를 삭제

2. LSTM(Long Short-Term Memory)

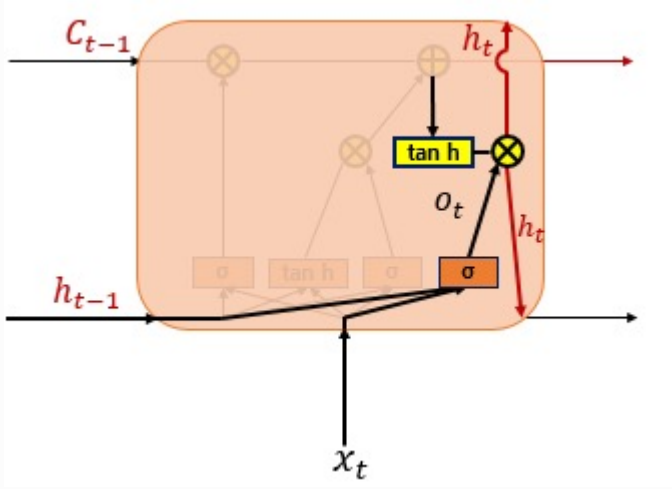
- **셀 상태(Long-Term)**



$$C_{t}=f_{t}\circ C_{t-1}+i_{t}\circ g_{t}$$

- 입력게이트 연산값(i,g)에 대한 곱연산 + 삭제게이트 연산값
- 입력데이터에 대해 이전 상태의 연산값의 기억 중요도에 따라 적용여부를 개별 판단

- **출력 게이트와 은닉상태 (Short-Term)**

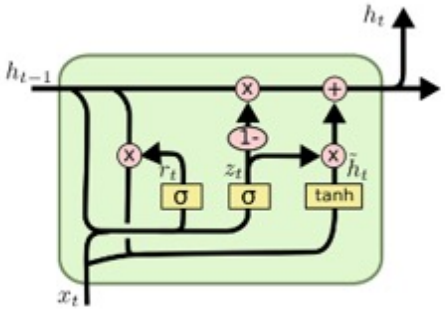


$$o_{t}=\sigma(W_{xo}x_{t}+W_{ho}h_{t-1}+b_{o})$$
$$h_{t}=o_{t}\circ\tanh(c_{t})$$

- t 에서의 x 값과 은닉상태에서의 결과갑을 합산
- 은닉상태는 망각게이트에 의해 걸러진 데이터가 잔류하는 현상태에서 유효한 데이터

3. 게이트 순환 유닛(Gated Recurrent Unit, GRU)

- **셀 상태(Long-Term)**

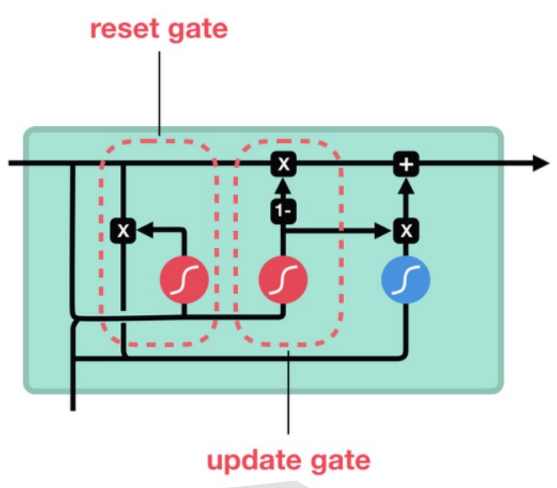


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \quad \text{---(1)}$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \quad \text{---(2)}$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \quad \text{---(3)}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad \text{---(4)}$$



o Reset Gate

- 이전 상태의 hidden state와 현재 상태의 x를 sigmoid 처리
- 이전 hidden state의 값 반영율 값

o Update Gate

- 이전 상태의 hidden state와 현재 상태의 x를 받아 sigmoid 처리
- 이전 정보와 현재정보의 반영비율 계산.
(= LSTM의 forget gate + input gate)
- 결과값은 다음 상태의 hidden state로 전송

4. LSTM vs GRU

- **GRU는 LSTM을 구성하는 Time-step의 Cell을 간소화한 버전**

- **차이점**

  - GRU는 LSTM과 다르게 Gate가 2개이며, Reset Gate(r)과 Update Gate(z)로 이룸

    > Reset Gate: 이전 상태 반영 비율

    > Update Gate: 이전 상태와 현재 상태의 반영 비율

  - LSTM에서의 Cell State와 Hidden State가 Hidden State로 통합

  - Update Gate가 LSTM에서의 forget gate, input gate를 제어

  - GRU에는 Output Gate가 없음

- **입력 데이터 양에 따라 성능 차이 발생**

  - 데이터 양이 적을 때는, 매개 변수의 양이 적은 GRU가 우수

  - 데이터 양이 더 많으면 LSTM이 우수

Chapter 3. Keras 기반 구현

Keras의 SimpleRNN과 LSTM 이해하기

1. 임의 입력 생성

```
train_X = [[[0.1, 4.2, 1.5, 1.1, 2.8], [1.0, 3.1, 2.5, 0.7, 1.1], [0.3, 2.1, 1.5, 2.1, 0.1], [2.2, 1.4, 0.5, 0.9, 1.1]]]
train_X = np.array(train_X, dtype=np.float32)
print(train_X.shape)
```

```
(1, 4, 5)
```

(batch_size, timesteps, input_dim)

예제에서 SimpleRNN과 LSTM은 재선언하여 사용 예정

- 값 자체 매번 초기화
- 이전 출력과 일관성 결여
- 출력값보다 **shape** 주목

1. 임의 입력 생성

return_sequences

False
(default)

True

return_state

2. SimpleRNN

은닉 상태의 크기

```
1 rnn = SimpleRNN(3)
2 # rnn = SimpleRNN(3, return_sequences=False, return_state=False)와 동일
3 hidden_state = rnn(train_X)
4
5 print('hidden state : {}, shape: {}'.format(hidden_state, hidden_state.shape))
```

마지막 시점의 은닉 상태만 출력

```
hidden state : [[ 0.985785   -0.88104105 -0.81002146]], shape: (1, 3)
```

마지막 시점의 은닉 상태 값 출력

2. SimpleRNN

```
1 rnn = SimpleRNN(3, return_sequences=True)
2 hidden_states = rnn(train_X)
3
4 print('hidden states : {}, shape: {}'.format(hidden_states, hidden_states.shape))
```

```
hidden states : [[[-0.99961203 -0.9970132  -0.9994677 ]
  [-0.9989546  -0.9482932  -0.95823294]
  [-0.9673317   0.6647892   0.5112487 ]
  [-0.96786255 -0.90565556  0.09610192]]], shape: (1, 4, 3)
```

모든 시점에 대한 은닉 상태 값 출력
(4 : timesteps)

2. SimpleRNN

```
1 rnn = SimpleRNN(3, return_sequences=True, return_state=True)
2 hidden_states, last_state = rnn(train_X)
3
4 print('hidden states : {}, shape: {}'.format(hidden_states, hidden_states.shape))
5 print('last hidden state : {}, shape: {}'.format(last_state, last_state.shape))
```

```
hidden states : [[[0.95856124 0.9106753  0.90559787]
  [0.9994177  0.9366319  0.9896112 ]
  [0.98607594 0.9853072  0.94421244]
  [0.99639845 0.9953609  0.9257922 ]]], shape: (1, 4, 3)
last hidden state : [[0.99639845 0.9953609  0.9257922 ]], shape: (1, 3)
```

return_sequence=True : 모든 시점의 은닉 상태 출력

return_state=True : 마지막 시점의 은닉 상태 출력

2. SimpleRNN

```
1 rnn = SimpleRNN(3, return_sequences=False, return_state=True)
2 hidden_state, last_state = rnn(train_X)
3
4 print('hidden state : {}, shape: {}'.format(hidden_state, hidden_state.shape))
5 print('last hidden state : {}, shape: {}'.format(last_state, last_state.shape))
```

```
hidden state : [[-0.93983185 -0.98487616 -0.84019357]], shape: (1, 3)
last hidden state : [[-0.93983185 -0.98487616 -0.84019357]], shape: (1, 3)
```

마지막 시점의 은닉 상태 출력

3. LSTM

```
1 lstm = LSTM(3, return_sequences=False, return_state=True)
2 hidden_state, last_state, last_cell_state = lstm(train_X)
3
4 print('hidden state : {}, shape: {}'.format(hidden_state, hidden_state.shape))
5 print('last hidden state : {}, shape: {}'.format(last_state, last_state.shape))
6 print('last cell state : {}, shape: {}'.format(last_cell_state, last_cell_state.shape))
```

```
hidden state : [[ 0.47393396  0.16043207 -0.20113361]], shape: (1, 3)
last hidden state : [[ 0.47393396  0.16043207 -0.20113361]], shape: (1, 3)
last cell state : [[ 0.7292785   0.371188   -0.63142097]], shape: (1, 3)
```

cell 상태까지 반환

3. LSTM

```
lstm = LSTM(3, return_sequences=True, return_state=True)
hidden_states, last_hidden_state, last_cell_state = lstm(train_X)

print('hidden states : {}, shape: {}'.format(hidden_states, hidden_states.shape))
print('last hidden state : {}, shape: {}'.format(last_hidden_state, last_hidden_state.shape))
print('last cell state : {}, shape: {}'.format(last_cell_state, last_cell_state.shape))
```

```
hidden states : [[[ 0.57998824 -0.05883592  0.4037268 ]
  [ 0.62203354  0.0098081   0.5812417 ]
  [ 0.624179   -0.13461015  0.62789434]
  [ 0.5782708   0.12843399  0.6389835 ]]], shape: (1, 4, 3)
last hidden state : [[0.5782708  0.12843399 0.6389835 ]], shape: (1, 3)
last cell state : [[1.6499138  0.20424938 2.2754965 ]], shape: (1, 3)
```

return_sequence=True : 모든 시점의 은닉 상태 출력

return_state=True : 마지막 시점의 은닉 상태 출력

Chapter 4. RNN 활용

RNNLM, Text Generation using RNN, Char RNN

1. RNNLM

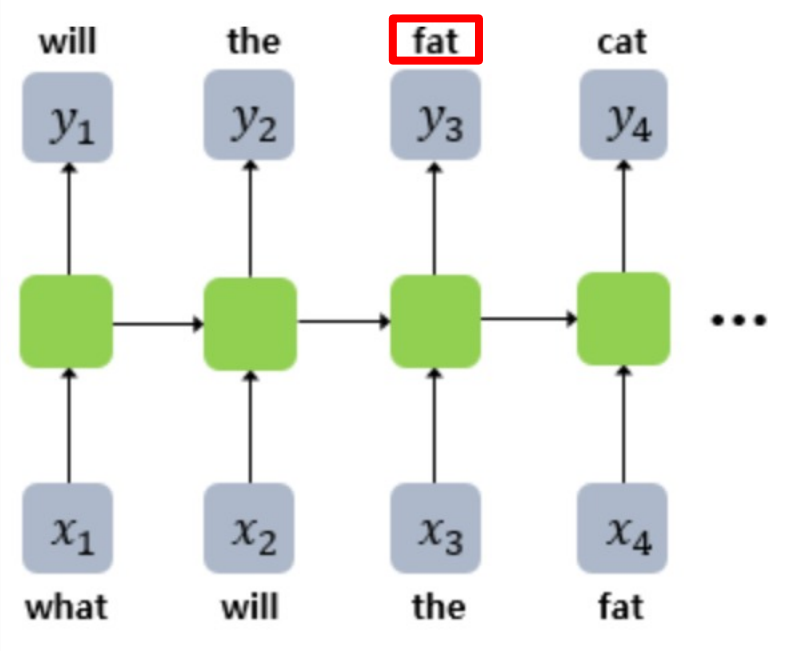
RNN 언어 모델
(Recurrent Neural Network Language Model)

시점 개념 도입으로 입력 길이 고정 불필요

“What will the fat cat sit on”

1. RNNLM

what, will, the라는 시퀀스로 인해 결정된 단어

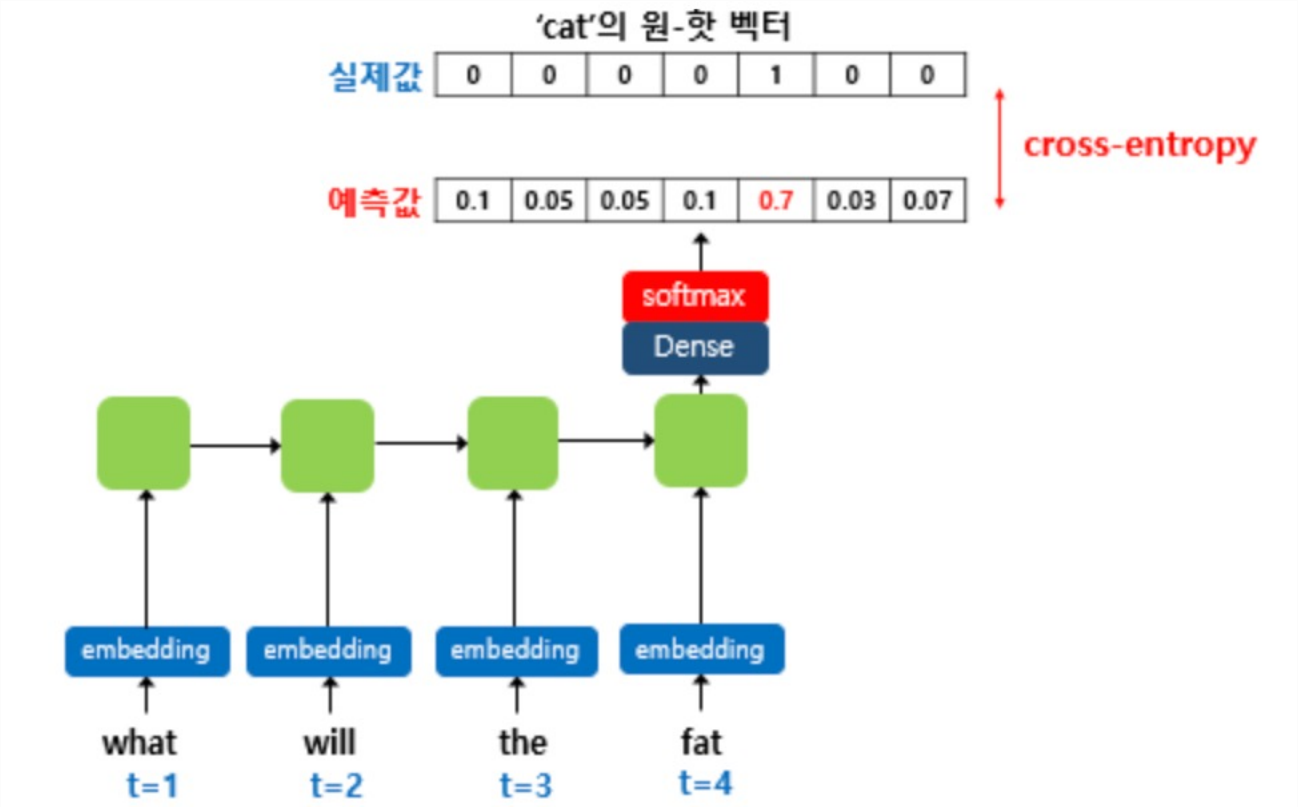


교사 강요
(Teacher Forcing)

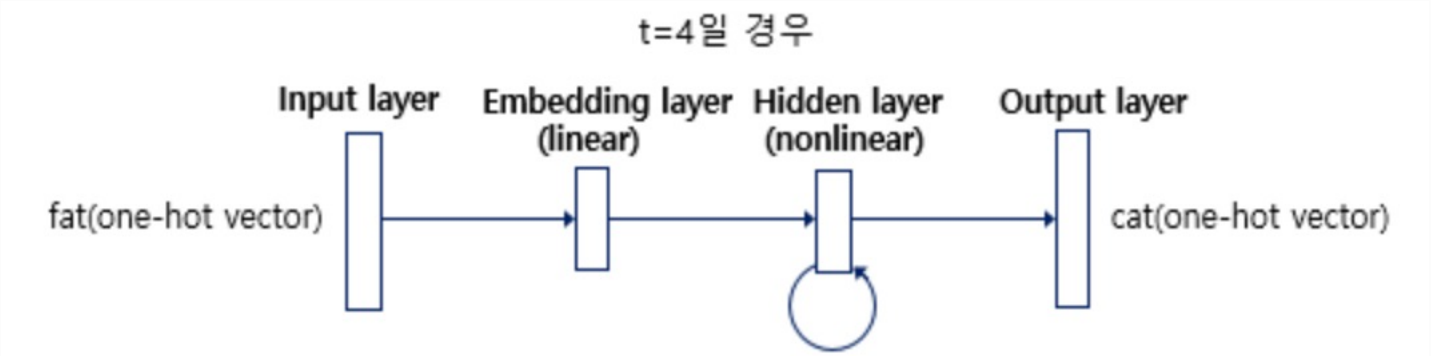
- 테스트 과정에서 t 시점의 출력이 t+1 시점의 입력으로 사용되는 RNN 모델 훈련 기법

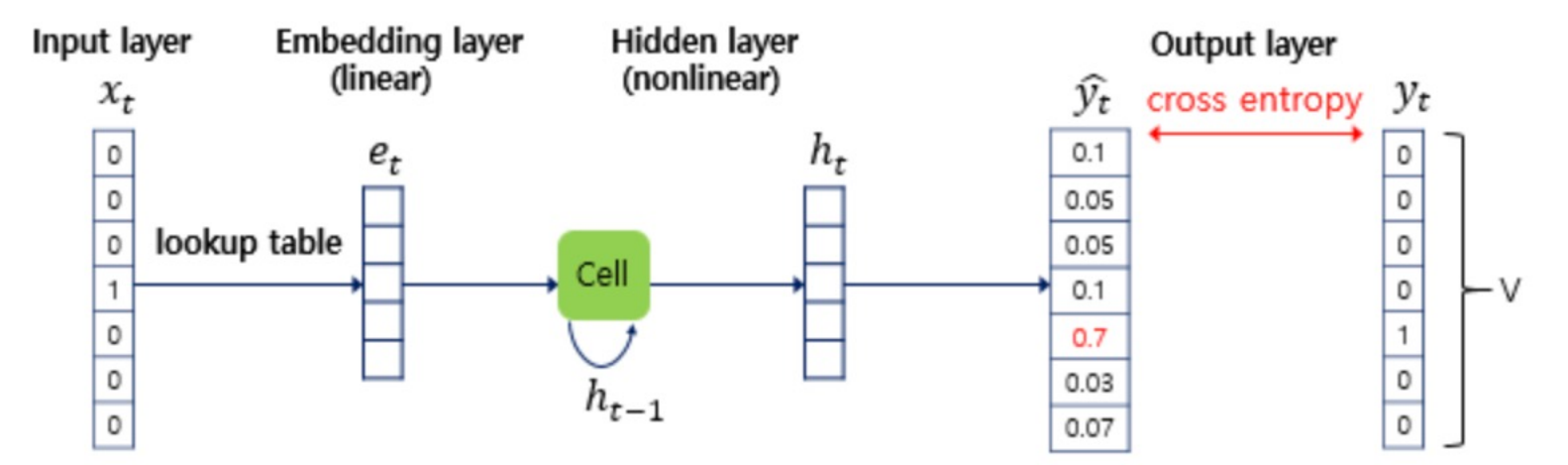
- t 시점에서의 예측값이 아닌 실제 알고있는 t시점의 레이블을 t+1 시점의 입력으로 사용

1. RNNLM

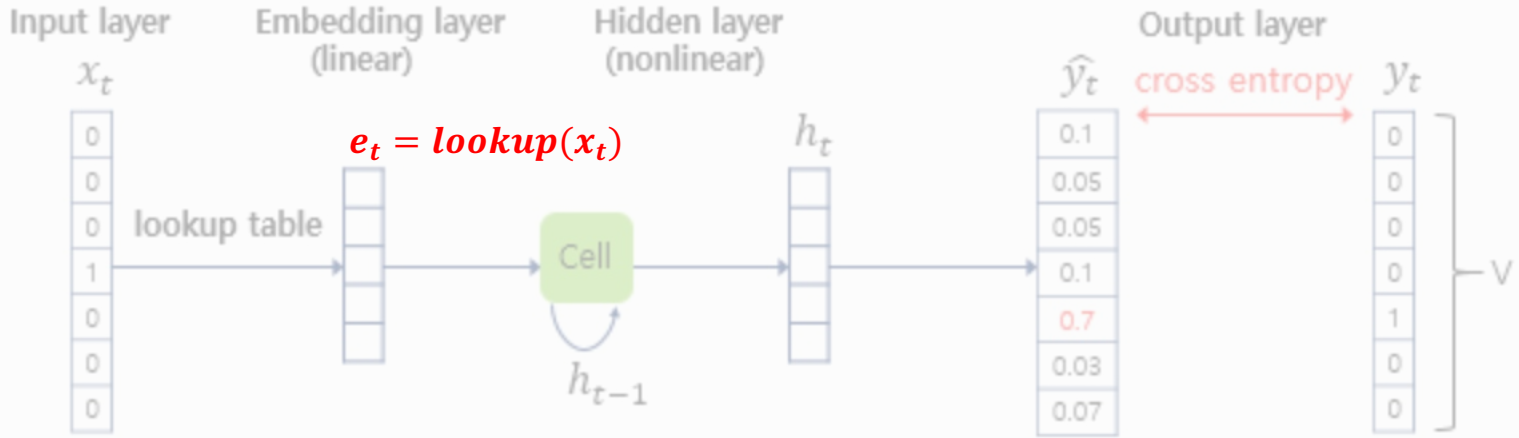

'cat'의 원-핫 벡터
실제값
0 0 0 0 1 0 0
cross-entropy
예측값
0.1 0.05 0.05 0.1 0.7 0.03 0.07
softmax
Dense
embedding
embedding
embedding
embedding
what
t=1
will
t=2
the
t=3
fat
t=4

1. RNNLM

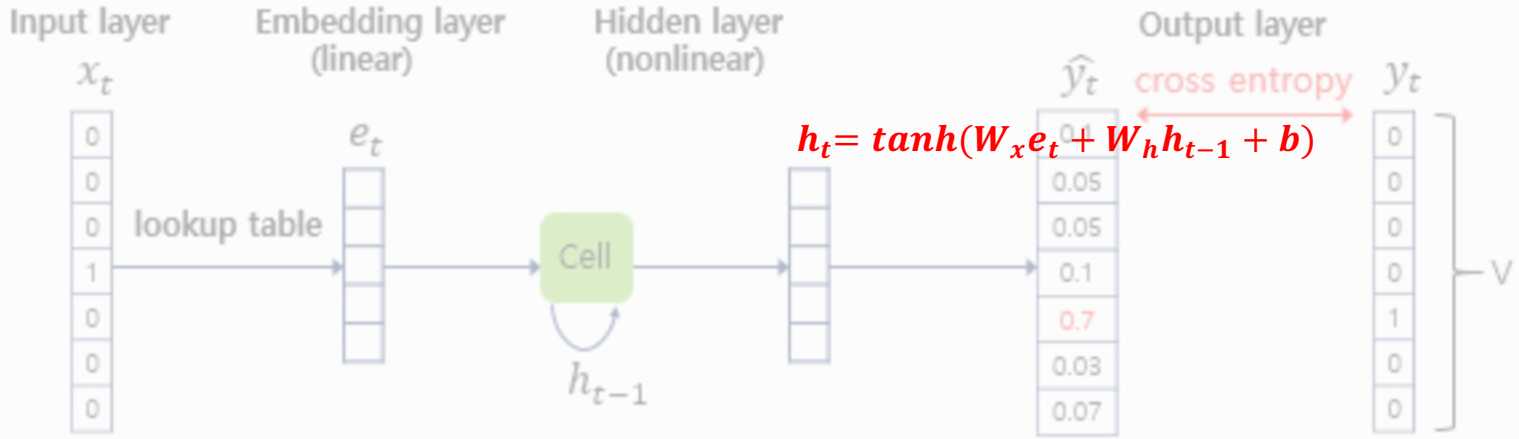

t=4일 경우
Input layer
Embedding layer (linear)
Hidden layer (nonlinear)
Output layer
fat(one-hot vector)
cat(one-hot vector)


Input layer
Embedding layer (linear)
Hidden layer (nonlinear)
Output layer
x_t
e_t
h_t
$\hat{y}_t$
cross entropy
y_t
lookup table
Cell
h_{t-1}
0.1
0.05
0.05
0.1
0.7
0.03
0.07
V

1. RNNLM


Input layer
x_t
Embedding layer (linear)
Hidden layer (nonlinear)
Output layer
$\hat{y}_t$
cross entropy
y_t
$e_t = lookup(x_t)$
h_t
lookup table
Cell
h_{t-1}
0
0
0
1
0
0
0
0.1
0.05
0.05
0.1
0.7
0.03
0.07
0
0
0
0
1
0
0
V

1. RNNLM


Input layer
Embedding layer (linear)
Hidden layer (nonlinear)
Output layer
x_t
e_t
$\hat{y}_t$
cross entropy
y_t
$h_t = tanh(W_x e_t + W_h h_{t-1} + b)$
lookup table
Cell
h_{t-1}
0
0
0
1
0
0
0
0.05
0.05
0.1
0.7
0.03
0.07
0
0
0
0
1
0
0
V

1. RNNLM

$$\hat{y}_t = softmax(W_y h_t + b)$$

- 각 차원 안에서의 값
- j번째 단어가 다음 단어일 확률
- 원-핫 벡터의 값에 근사해야 함
- 역전파가 이루어지면서 가중치 행렬 학습

  → Embedding vector값 학습

2. RNN을 이용하여 텍스트 생성하기

Tokenizer

Making Training set (padding)

Training with RNN

2. RNN을 이용하여 텍스트 생성하기

```
1 print(sentence_generation(model, t, '경마장에', 4))
```

```
경마장에 있는 말이 뛰고 있다
```

```
1 print(sentence_generation(model, t, '그의', 2))
```

```
그의 말이 법이다
```

```
1 print(sentence_generation(model, t, '가는', 5))
```

```
가는 말이 고와야 오는 말이 곱다
```

```
1 print(sentence_generation(model, t, '가는', 10))
```

```
가는 말이 고와야 오는 말이 곱다 뛰고 있다 곱다 있다 곱다
```

3. LSTM을 이용하여 텍스트 생성하기

Preprocessing

Tokenizer

Making Training set

Training with LSTM

3. LSTM 을 이용하여 텍스트 생성하기

Preprocessing

- Unknown 제거
- 문장부호 제거, 소문자
- 정수인코딩
- pre-padding

3. LSTM을 이용하여 텍스트 생성하기

Preprocessing

Word embedding

Text normalization

3. LSTM을 이용하여 텍스트 생성하기

Training with LSTM

LSTM(128)

Dense with softmax

3. LSTM을 이용하여 텍스트 생성하기

```
1 print(sentence_generation(model, t, 'i', 10))
```

```
i cant jump ship from facebook yet advice for house and
```

```
1 print(sentence_generation(model, t, 'how', 10))
```

```
how can a doctor grapple with the epidemic of cost assassination
```

4. Char RNNLM을 이용하여 텍스트 생성하기

Preprocessing

Tokenizer

Making Training set (padding)

Training with RNN

4. Char RNNLM을 이용하여 텍스트 생성하기

Preprocessing

\r, \n 제거

소문자

정수 인코딩

pre-padding

4. Char RNNLM을 이용하여 텍스트 생성하기

Preprocessing

Training with LSTM

LSTM(256)

LSTM(256)

Timedistributed Dense

4. Char RNNLM을 이용하여 텍스트 생성하기

Timedistributed Dense

Consider a batch of 32 samples, where each sample is a sequence of 10 vectors of 16 dimensions. The batch input shape of the layer is then (32, 10, 16), and the input_shape, not including the samples dimension, is (10, 16).

```
# as the first layer in a model
model = Sequential()
model.add(TimeDistributed(Dense(8), input_shape=(10, 16)))
# now model.output_shape == (None, 10, 8)
```

The output will then have shape (32, 10, 8).

4. Char RNNLM을 이용하여 텍스트 생성하기

```
1 print(sentence_generation(model, char_to_index, 10, 'I get on w', 80))
```

```
I get on with life as a programmer, I like to contemplate beer. But when I stop my talking
```

감사합니다.