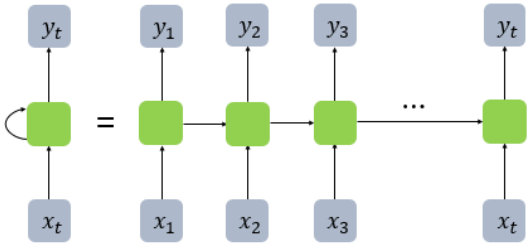
RNN

순환 신경망(Recurrent Neural Network, RNN)

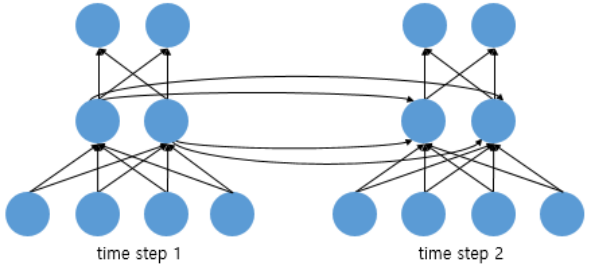
RNN은 입력과 출력을 시퀀스 단위로 처리하는 **시퀀스(Sequence) 모델**

RNN은 은닉층의 노드에서 활성화 함수를 통해 나온 결과값을 출력층 방향으로도 보내면서, **다시 은닉층 노드의 다음 계산의 입력**으로 보내는 특징이 있음

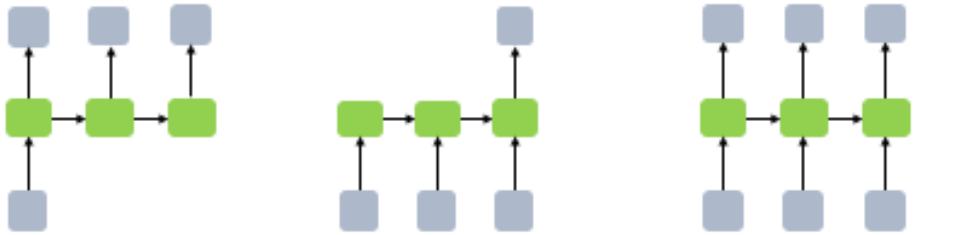
이 셀은 메모리 셀 혹은 RNN 셀이라 함



메모리 셀이 출력층 방향 또는 다음 시점인 t+1의 자신에게 보내는 값을 **은닉 상태(hidden state)** 라고 함



순환 신경망(Recurrent Neural Network, RNN)

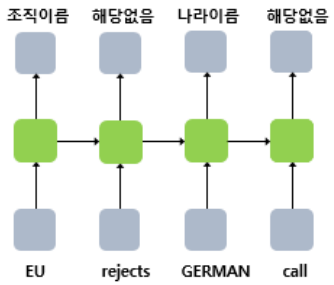


일 대 다(one-to-many) 다 대 일(many-to-one) 다 대 다(many-to-many)

입력과 출력의 길이에 따라서 RNN은 다양한 형태를 가짐

예)
- **일대다**: 이미지→사진 제목(단어들의 나열)
- **다대일**: 감성 분류, 스팸 메일 분류 등 텍스트 분류
- **다대다**: 챗봇, 번역기, 개체명 인식, 품사 태깅 등

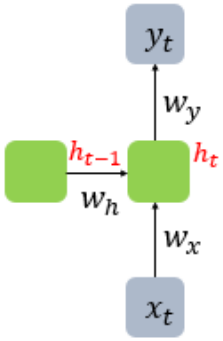
스팸 메일 분류



개체명 인식

순환 신경망(Recurrent Neural Network, RNN)

RNN의 수식



- 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.

Ex. 은닉층 연산 $d = 4, D_h = 4$

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

x_t $(d \times 1)$는 단어 벡터
d는 단어 벡터의 차원
W_x $(D_h \times d)$는 입력층을 위한 가중치
D_h는 은닉 상태 크기
W_h $(D_h \times D_h)$는 은닉 상태값을 위한 가중치
$h_{t-1}(D_h \times 1)$는 t-1의 은닉 상태값
b $(D_h \times 1)$는 편향

순환 신경망(Recurrent Neural Network, RNN)

Keras로 RNN 구현

```
from tensorflow.keras.layers import SimpleRNN

model.add(SimpleRNN(hidden_units))
```

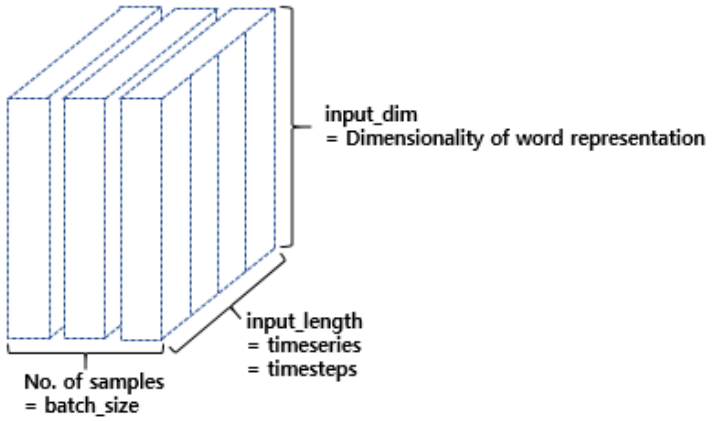
```
# 추가 인자를 사용할 때
model.add(SimpleRNN(hidden_units, input_shape=(timesteps, input_dim)))

# 다른 표기
model.add(SimpleRNN(hidden_units, input_length=M, input_dim=N))
```

hidden_units = 은닉 상태의 크기를 정의. 메모리 셀이 다음 시점의 메모리 셀과 출력층으로 보내는 값의 크기(output_dim)와도 동일. RNN의 용량(capacity)을 늘린다고 보면 되며, 중소형 모델의 경우 보통 128, 256, 512, 1024 등의 값을 가짐

timesteps = 입력 시퀀스의 길이(input_length)라고 표현하기도 함. 시점의 수.

input_dim = 입력의 크기.



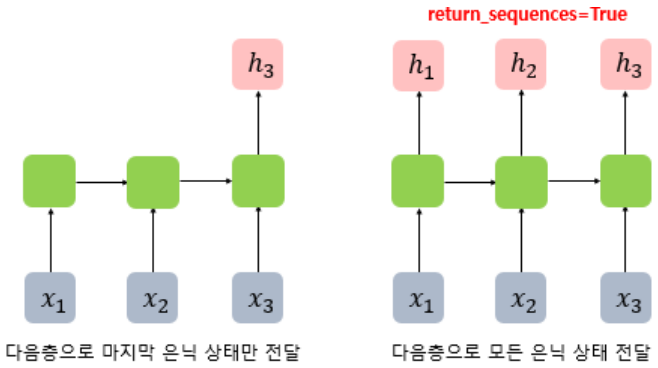
RNN 층은 (batch_size, timesteps, input_dim) 크기의 3D 텐서를 입력으로 받음.

batch_size는 한 번에 학습하는 데이터의 개수를 말함

주의할 점은 위 코드는 출력층까지 포함한 인공 신경망 코드가 아니라 주로 은닉층으로 간주할 수 있는 하나의 RNN 층에 대한 코드

순환 신경망(Recurrent Neural Network, RNN)

Keras로 RNN 구현



return_sequences=True를 선택하면 메모리 셀이 모든 시점(time step)에 대해서 은닉 상태값을 출력 → (batch_size, timesteps, output_dim) 크기의 3D 텐서를 리턴, (다대다)

별도 기재하지 않거나 return_sequences=False로 선택할 경우에는 메모리 셀은 하나의 은닉 상태값만을 출력 → (batch_size, output_dim) 크기의 2D 텐서를 리턴 (다대일)

```
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
_________________________________________________________________
```

출력값이 (batch_size, output_dim) 크기의 2D 텐서일 때,
output_dim은 hidden_units의 값인 3.
batch_size를 현 단계에서는 알 수 없으므로 (None, 3)이 됨

순환 신경망(Recurrent Neural Network, RNN)

Keras로 RNN 구현

```
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
_________________________________________________________________
```

batch_size를 미리 8로 정의하면 출력의 크기가 (8, 3)이 됨

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

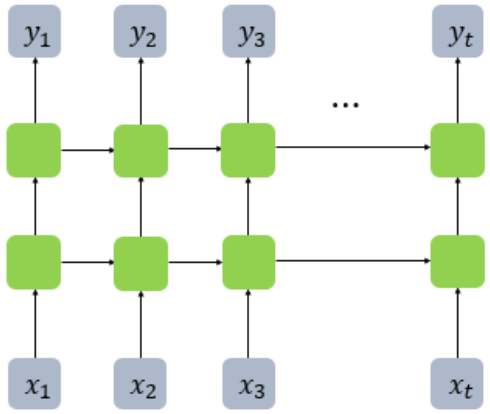
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_3 (SimpleRNN)     (8, 2, 3)                 42
=================================================================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

return_sequences=True를 기재하여 출력값으로 (batch_size, timesteps, output_dim) 크기의 3D 텐서를 리턴하도록 모델을 만들면, 출력의 크기가 (8, 2, 3)이 됨

순환 신경망(Recurrent Neural Network, RNN)

Deep RNN

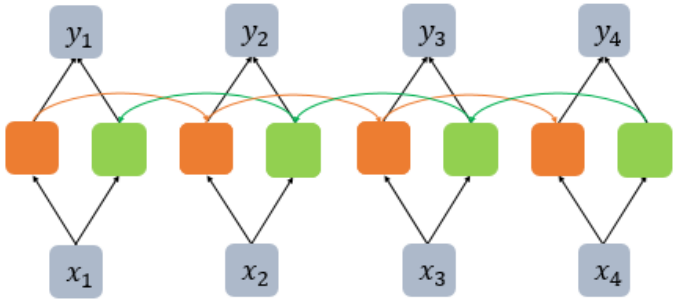
ex. 은닉층이 여러개




```
model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=32, input_length=100))
model.add(SimpleRNN(units=32, return_sequences=True))
model.add(SimpleRNN(units=32, return_sequences=True))
model.add(SimpleRNN(units=32, return_sequences=True))
model.add(SimpleRNN(units=32))
model.summary()
```

네트워크의 표현력을 증가시키기 위해 여러 개의 순환 층을 차례대로 쌓는 것이 유용할 때가 있음

양방향 RNN(Bidirectional RNN)




```
from tensorflow.keras.layers import Bidirectional

timesteps = 10
input_dim = 5

model = Sequential()
model.add(Bidirectional(SimpleRNN(hidden_units, return_sequences=True), input_shape=(timesteps, i
nput_dim)))
```

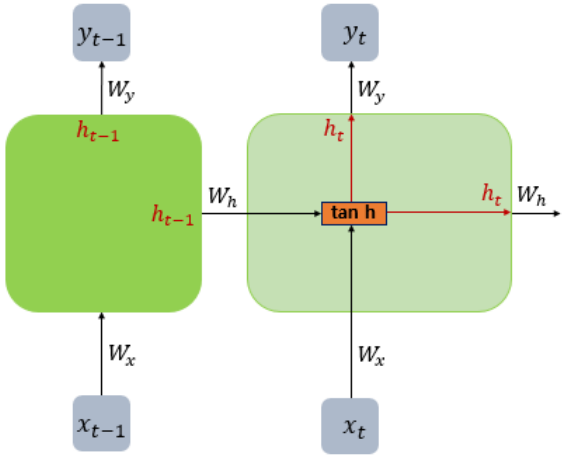
양방향 RNN은 하나의 출력값을 예측하기 위해 기본적으로 두 개의 메모리 셀을 사용.

- 첫번째 메모리 셀은 앞 시점의 은닉 상태(Forward States) 를 전달받아 현재의 은닉 상태를 계산.
- 두번째 메모리 셀은 뒤 시점의 은닉 상태(Backward States) 를 전달받아 현재의 은닉 상태를 계산

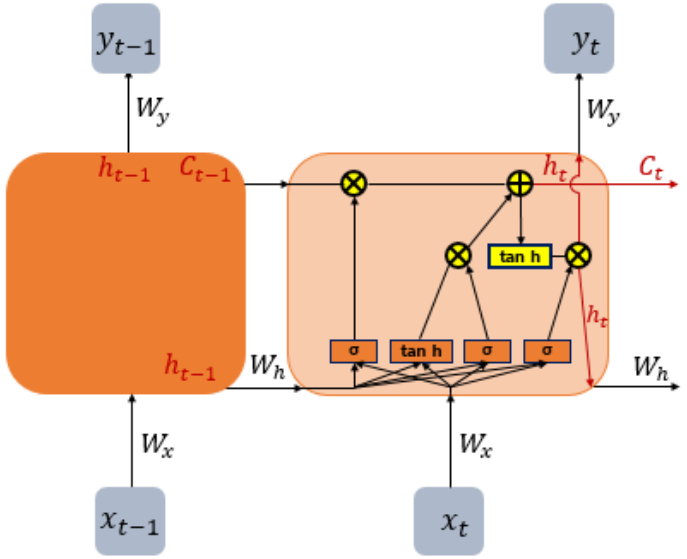
장단기 메모리(Long Short-Term Memory, LSTM)

기존 RNN은 시점(time step)이 길어질 수록 앞의 정보가 뒤로 충분히 전달되지 못하는 현상이 발생 (장기 의존성 문제(the problem of Long-Term Dependencies)

LSTM은 은닉층의 메모리 셀에 **입력 게이트**, **망각 게이트**, **출력 게이트**를 추가하여 불필요한 기억을 지우고, 기억해야 할 것들을 정함

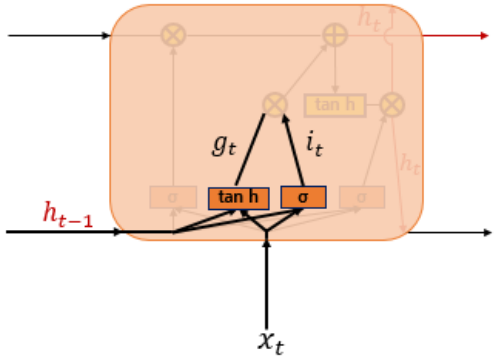


기본 RNN



LSTM의 전체적인 내부 모습

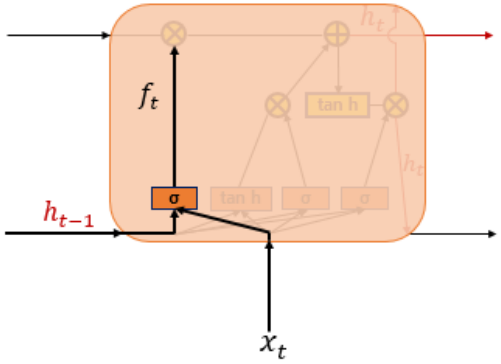
장단기 메모리(Long Short-Term Memory, LSTM)



입력 게이트: 현재 정보를 기억하기 위한 게이트

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$g_t = tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$

시그모이드 함수 -> 0, 1사이 값
하이퍼볼릭탄젠트 함수 -> -1, 1 사이 값
→ 선택된 기억할 정보의 양을 정함

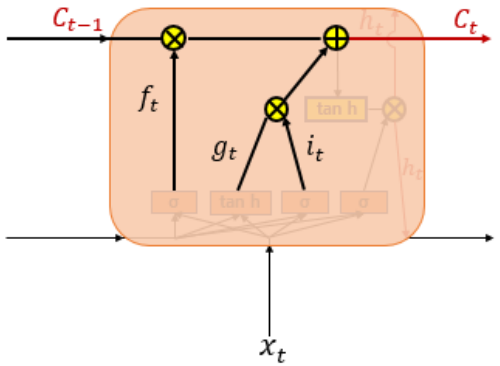


삭제 게이트: 기억을 삭제하기 위한 게이트

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

시그모이드 함수 -> 0, 1사이 값
→삭제 과정을 거친 정보의 양
- 0에 가까울수록 정보가 많이 삭제,
- 1에 가까울수록 정보를 온전히 기억

장단기 메모리(Long Short-Term Memory, LSTM)

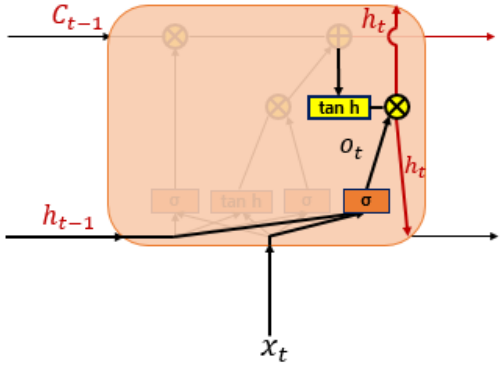


셀 상태

$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$

- 입력 게이트에서 구한 i_t, g_t 의 원소별 곱 → 기억할 값
- C_{t-1} , f_t 의 원소별 곱 → 삭제 게이트 결과 값

C_t 는 t시점의 셀 상태로, t+1 시점의 LSTM 셀로 넘겨짐



출력 게이트와 은닉 상태

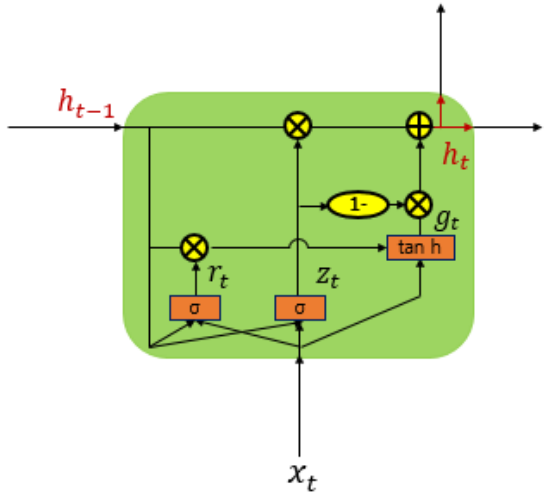
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$h_t = o_t \circ tanh(c_t)$$

출력게이트 o_t는 현재 시점의 은닉 상태를 결정

은닉상태 값 h_t는 또한 출력층으로 향함

게이트 순환 유닛(Gated Recurrent Unit, GRU)

GRU는 LSTM의 장기 의존성 문제에 대한 해결책을 유지하면서, 은닉 상태를 업데이트하는 계산을 줄임
GRU는 성능은 LSTM과 유사하면서 복잡했던 LSTM의 구조를 **간단화**



GRU는 **업데이트 게이트**와 **리셋 게이트** 두 가지 게이트만이 존재

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$g_t = tanh(W_{hg}(r_t \circ h_{t-1}) + W_{xg}x_t + b_g)$$
$$h_t = (1 - z_t) \circ g_t + z_t \circ h_{t-1}$$

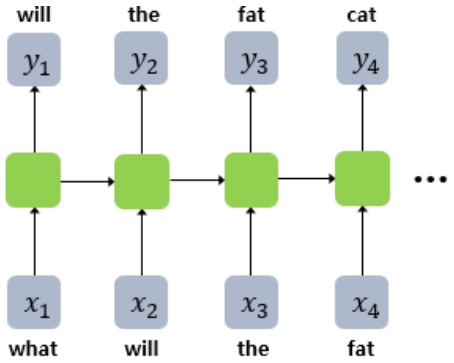
업데이트 게이트는 LSTM의 Forget gate와 Input gate의 역할을 모두 담당

- z_t가 이전 정보의 비율을 결정하고 $1 - z_t$가 여기에 대응되는 현재 정보의 비율을 결정.
- 즉 과거의 정보를 얼마만큼 가져가고 현재 새로운 정보를 얼마만큼 가져갈지를 정함.
- z_t는 forget gate, $1 - z_t$는 input gate의 역할.

리셋 게이트 (r_t)는 이전 정보에서 얼마만큼을 선택해서 내보낼지를 결정

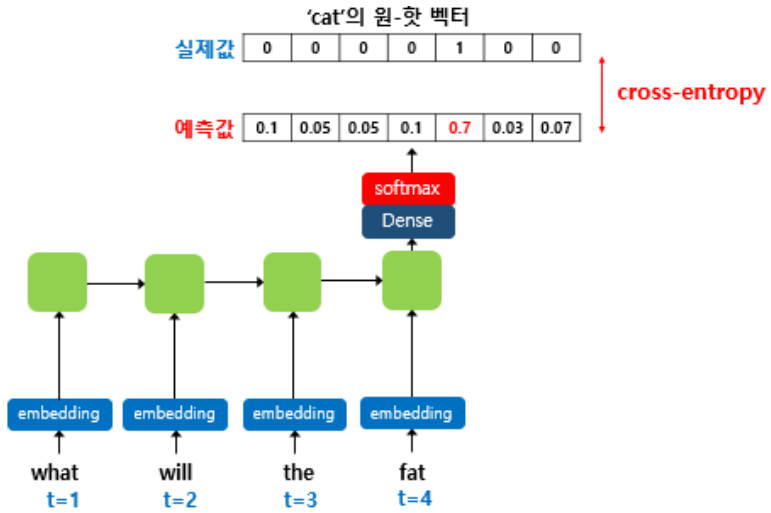
- LSTM의 output gate와 유사한 역할.
- LSTM의 output gate는 최종 출력에서 output으로 나갈 정보를 선별하는 것이라면 GRU의 reset gate는 이전 정보에서 미리 내보낼 정보를 선별

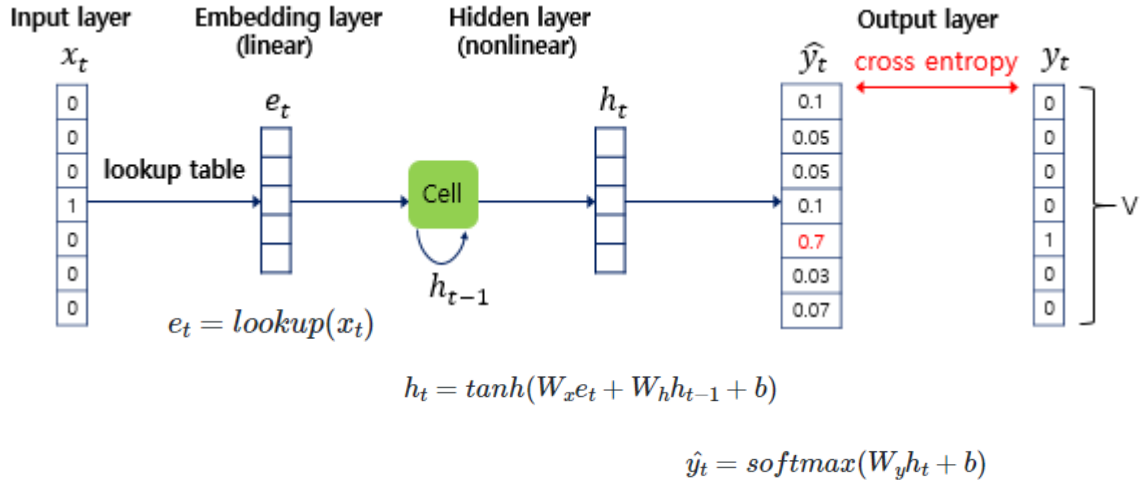
RNN 언어모델



RNNLM은 기본적으로 예측 과정에서 이전 시점의 출력을 현재 시점의 입력으로 함.
즉, 네번째 시점의 cat은 앞서 나온 what, will, the, fat이라는 시퀀스로 인해 결정된 단어임.

하지만, **훈련 과정**에서는 모델이 t 시점에서 예측한 값을 t+1 시점에 입력으로 사용하지 않고,
t 시점의 **레이블을** t+1 시점의 입력으로 사용 : 이를 **교사 강요(teacher forcing)**이라고 함





출력층의 활성화함수로 softmax, 손실 함수로 cross-entropy 사용.

참고자료

- 딥 러닝을 이용한 자연어 처리 입문(https://wikidocs.net/book/2155)
- Gated Recurrent Unit(GRU)의 이해(https://velog.io/@lighthouse97/Gated-Recurrent-UnitGRU%EC%9D%98-%EC%9D%B4%ED%95%B4)