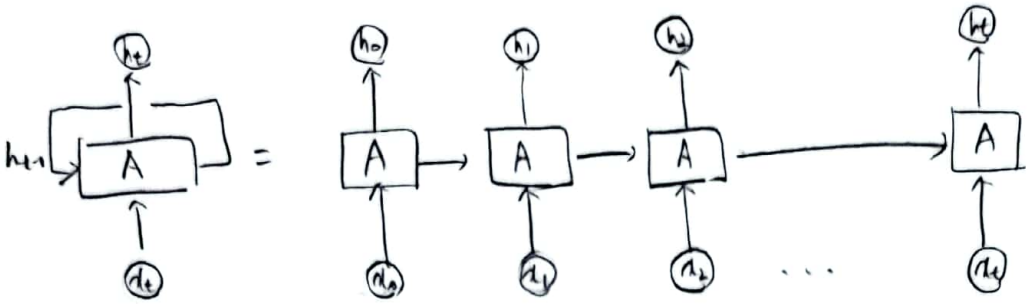
RNN (Recurrent Neural Network)

sequence가 입력 값으로 주어질 때,

각 time step에서 들어오는 입력벡터 x_t와 그 전 time step의 RNN 모듈에서 계산한 hidden state 벡터 h_{t-1}을 입력으로 받아 현재 time step에서의 h_t를 업데이트 해주는 구조



매 time step마다 동일한 파라미터를 가진 모듈을 사용

(A)

hidden state 계산 방법

- h_{t-1} : old hidden-state vector
- x_t : input vector at some time step
- h_t : new hidden-state vector
- f_W : RNN function with parameters W
- y_t : out put vector at time step t → 매 time step마다 계산하는 경우와 마지막 time step에서만 계산하는 경우로 나뉨

$$h_t = f_W(h_{t-1}, x_t)$$

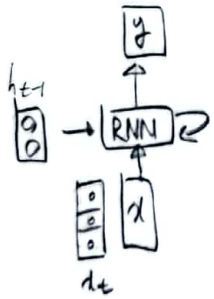
↳ 모든 time step에서 출력

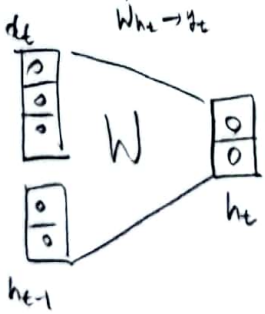
$$h_t = f_W(h_{t-1}, x_t) \rightarrow h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

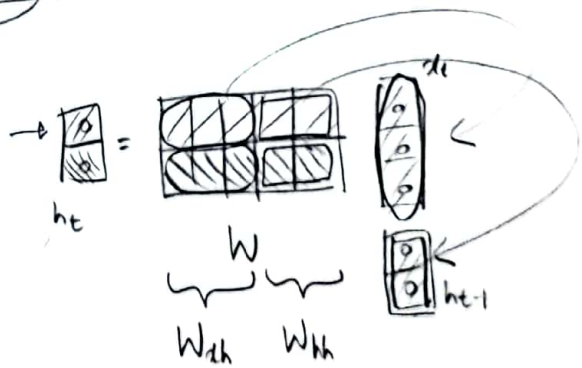
$$y_t = W_{hy} h_t$$

$W_{hh} \rightarrow y_t$

ex)







$$= W_{xh} x_t + W_{hh} h_{t-1}$$

$W_{xh} \rightarrow h_t$ $W_{hh} \rightarrow h_t$

y_t 이진분류 : scalar → sigmoid

다중분류 : vector → softmax

dim : # class

RNN 유형

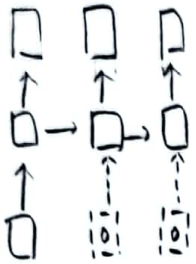
① One-to-One

ex) [키, 몸무게, 나이] → 러닝함, 안함!

Standard neural networks

입력, 출력 모두 sequence가 아님 (하나의 time step)

② One-to-many



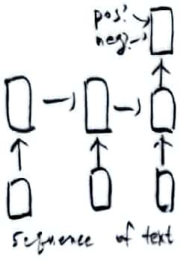
입력이 하나의 time step이고 출력은 여러 time step

ex) 이미지 캡셔닝

하나의 이미지를 입력으로 주면 이미지에 대한 설명문을 생성

입력이 첫번째 time step에만 들어감 (나머지 입력은 0으로 간주)

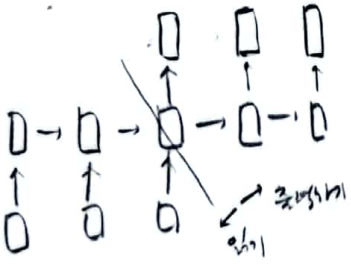
③ Many-to-one



입력은 sequence로 받아 매번 time step에서 출력

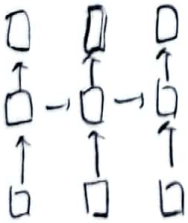
ex) 감정 분석

④ Many-to-many (Sequence-to-sequence)



입력과 출력 모두 sequence

ex) 기계 번역



ex) Video Classification on frame level

pos tagging.

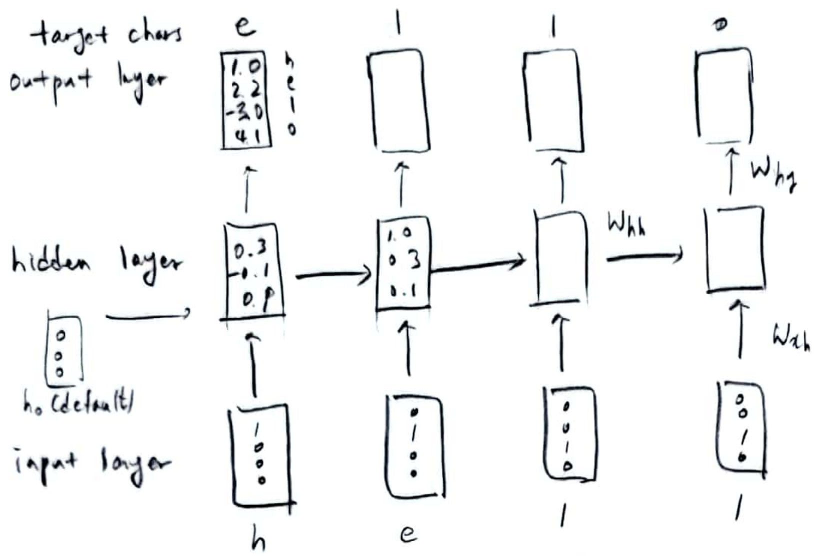
Character-level language model

언어모델 이전에 등장한 문자열을 기반으로 다음 단어를 예측하는 task

여기서는 문자 단위.

ex) "hello"가 주어져 있다고 하면

Vocabulary: {'h', 'e', 'l', 'o'}, $h: \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ $e: \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$



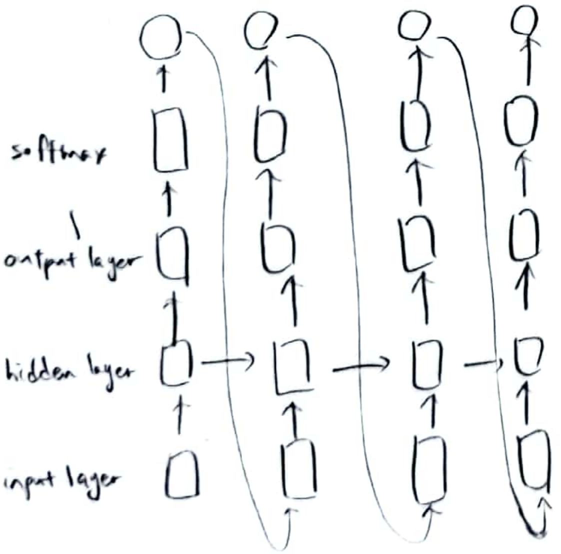
$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b)$

$Logit = W_{hy} h_t + b$

many-to-many

h가 주어지면 e, e가 주어지면 l, ··· 을 예측

test time 에서는



ex) 전 time step 까지의 주식값을 활용해서
다음 날 주식값 예측,
셰익스피어의 글 (원안),
인물별 대사
latex 논문,
C 프로그래밍.

BPTT(Backpropagation through time)

Loss function에 대한 backpropagation을 통해 W_{xh}, W_{hh}, W_{hy}를 학습

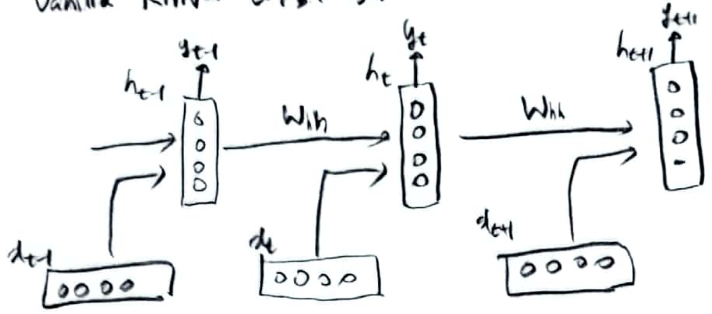
sequence가 길어지면 전체 sequence로 loss를 계산하고 gradient를 계산하는 것은 한계가 존재

truncation : 군데군데 잘라서 제한된 길이로 학습

특정 hidden state를 시각화하면 time step이 진행됨에 따라 중요한 부분을 잘 학습하는 걸 알 수 있다.

ex) If statement cell, quote detection cell...

Vanilla RNN의 문제점: 동일한 matrix를 매 time step마다 곱하는데



등비수열처럼되니, backpropagation에서도 마찬가지로 같은 수가 계속 곱해지므로

Vanishing Gradient 또는 Exploding Gradient가 발생

(공비 <1) (공비 >1)

예시) $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b)$, $t=1,2,3$

For $W_{hh}=3$, $W_{xh}=2$, $b=1$

$h_3 = \tanh(2x_3 + 3h_2 + 1)$

$h_2 = \tanh(2x_2 + 3h_1 + 1)$

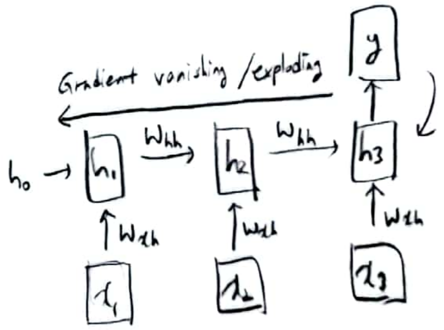
$h_1 = \tanh(2x_1 + 3h_0 + 1)$

$h_3 = \tanh(2x_3 + 3\tanh(2x_2 + 3\cdots + 1) + 1)$

3이 time step마다 계속 곱해짐 → exploding

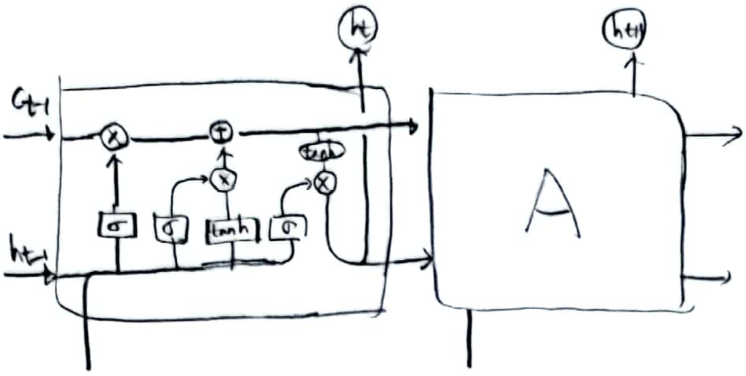
만약 $W_{hh} = 0.2$이면 → vanishing

(Long-term dependency 문제)



LSTM (Long Short-Term Memory)

RNN에서 hidden state vector는 단기 기억 소자로 보면 time step이 진행됨에 따라 단기기억을 보다 길게 기억할 수 있도록 개선한 모델



$$h_t = f_W(x_t, h_{t-1})$$

(RNN)

$$\{c_t, h_t\} = LSTM(x_t, c_{t-1}, h_{t-1})$$

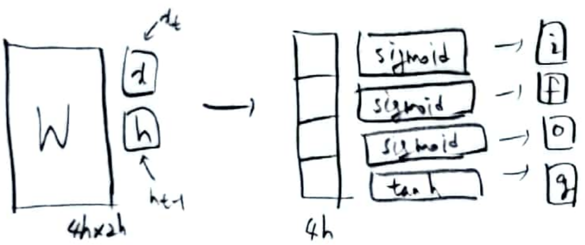
- c_t, h_t: 출력 산출할 정보 포함 / cell state는 계속 해당 time step에서 도출한 필요가 있는 정보만 남긴다
- c_{t-1}: cell state
- h_{t-1}: hidden state

i : input gate, cell에 쓸지 말지 결정, σ를 통해 0~1 범위의 값, cell/hidden state 두 가지로 오른다

f : forget gate, 정보를 어느정도 지울지, σ를 통해 0~1 범위의 값

o : output gate, cell 정보를 어느정도 hidden state에서 사용할지, σ를 통해 0~1 범위의 값

g : Gate gate, 어느정도로 cell state에 반영할지, tanh를 통해 -1~1 범위의 값



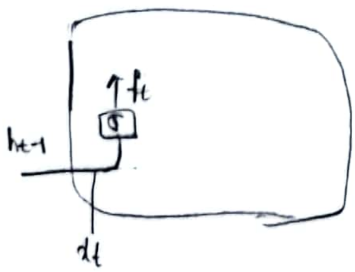
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot tanh(c_t)$$

IFOG는 전 time에서 넘어온 C_{t-1}는 적절하게 변환

Forget gate $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$



ex) $C_{t-1} = \begin{bmatrix} 3 \\ 5 \\ -2 \end{bmatrix}$

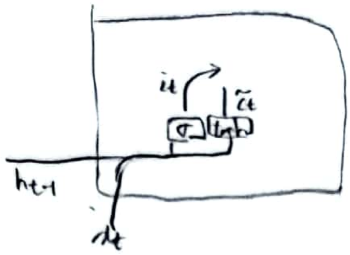
만약 $f_t = \begin{bmatrix} 0.7 \\ 0.4 \\ 0.8 \end{bmatrix}$이면

element-wise product를 통해 $\begin{bmatrix} 2.1 \\ 2.0 \\ -1.6 \end{bmatrix}$이 된다

Input gate : $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Gate gate $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

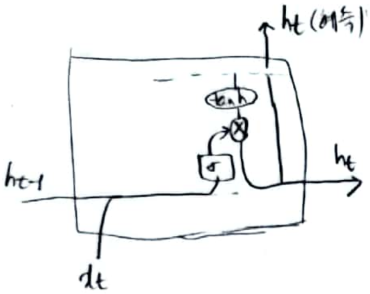
new cell state $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$



한 번의 선형변환만으로 C_{t-1}에 더해줄 정보를 만들기 어려우면 그거보다 좀 더 큰 값들 $\tilde{C}_t$로 만들어준 후 i_t를 이용해 일부만 반영해서, 실제로 더해주려 하는 정보를 2단계에 걸쳐서 만든다

output gate $o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$

hidden state $h_t = o_t \cdot \tanh(C_t)$



C_t와 h_t의 차이?

→ C_t: 기억해야하는 모든 정보를 담은 벡터.

h_t: 현재 time step에서 output layer의 입력으로 사용되는 벡터, 해당 time step의 예측값에 직접적으로 필요한 정보를 담음(필터링)

ex) "hell 에서

따옴표가 열려있는 것은 당장 필요한 정보는 아니지만 기억해 두고는 있어야 한다. 해당 정보는 C_t에 저장

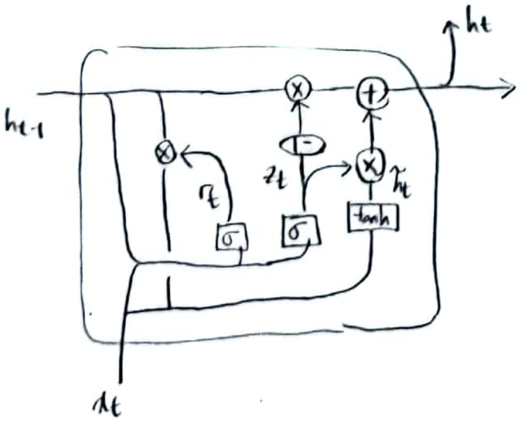
다음에 o가 와야할때는 당장 따옴표가 필요하지 않고 필터링한 부분을 h_t에 저장

GRU (Gated Recurrent Unit)

LSTM의 경량화. 적은 메모리, 빠른 계산시간

cell state와 hidden state를 hidden state로 일원화

따라서 hidden state는 LSTM의 cell state와 비슷한 역할



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

$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$

$\tilde{h}_t = \tanh(W [r_t \cdot h_{t-1}, x_t])$

$h_t = (1 - z_t) h_{t-1} + z_t \tilde{h}_t$ (가중평균)

cf) $C_t = f_t C_{t-1} + i_t \tilde{C}_t$ ← g

ex) $z_t = \begin{bmatrix} 0.6 \\ 0.3 \\ 0.8 \end{bmatrix}$ $1 - z_t = \begin{bmatrix} 0.4 \\ 0.7 \\ 0.2 \end{bmatrix}$

→ 가중치를 덧셈함으로써 gradient 복사 : Long-term dependency 해결.