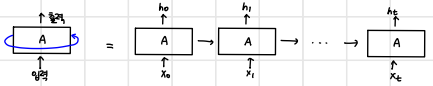
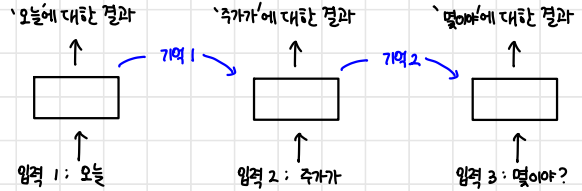
인공지능이 문장을 듣고 이해 (∵ 이미 학습시켜 놓았기 때문)



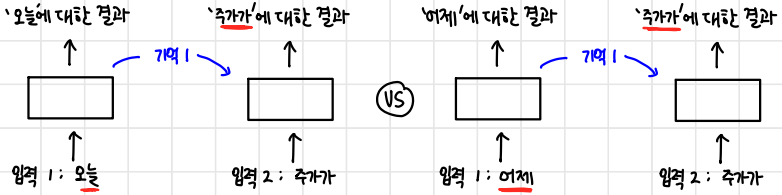
순서중요 by 순환신경망 (RNN)

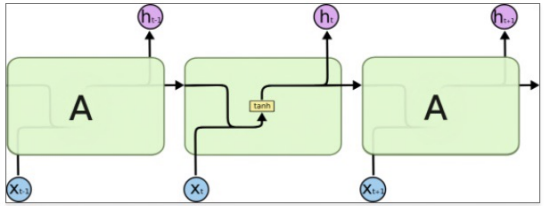
여러개의 데이터가 순서대로 입력되었을때 앞서 입력받은 데이터 장시기억,
기억된 데이터의 중요도를 판단해 가중치 줌

ex) "오늘 추가가 떨어야?"



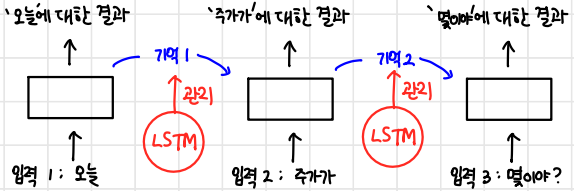
순환이 되는 가운데 앞서 나온 입력에 대한 결과가 뒤에 나오는 입력값에 영향을 줌
이래야 비슷한 문장이 입력되었을때 차이 구별하며 출력값에 반영 가능





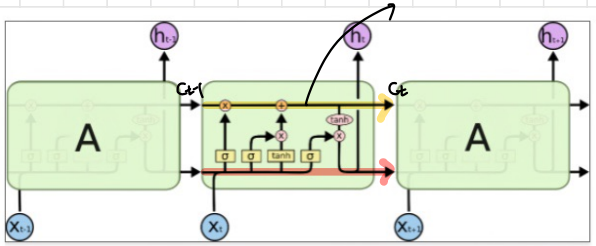
tanh layer을 모듈로 사용

- LSTM은 한 층 안에서 반복을 많이 해야하는 RNN의 특성상 일반신경망보다 기울기소실문제가 더 많이 발생하고 이를 해결하기 어렵다는 단점을 보완한 방법
  즉, 반복되기 직전에 다음 층으로 기억된 값을 얼마나 안 얼길지를 관리하는 단계 추가



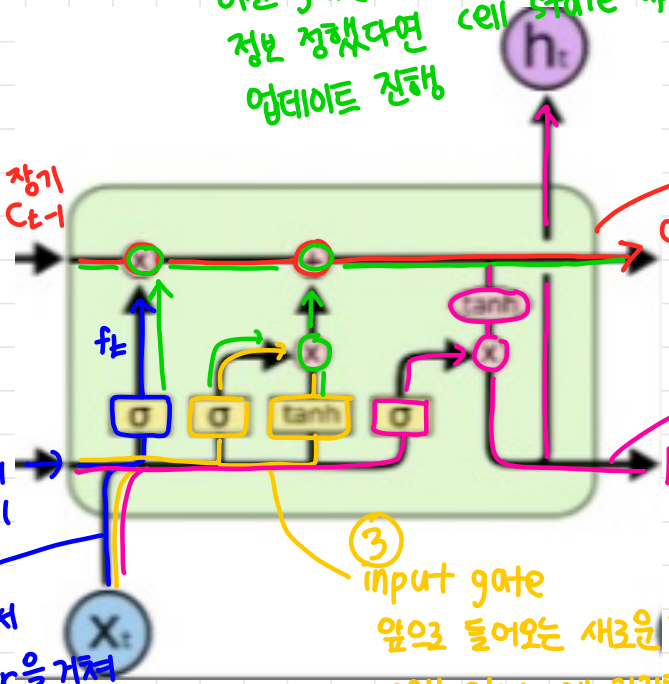
기억값의 가중치를 관리하는 장치

주로 시계열처리나 자연어처리에 사용됨



4개의 layer 가 서로 정보 주고받는 구조, 상태 < 단기상태 / 장기상태

④ cell state update
이전 gate에서 버릴 정보들과 업데이트할
정보 정했다면 cell state update 과정에서
업데이트 진행



① cell state
장기상태, cell state 정보가 바뀌지
않고 그대로 흐르도록

⑤ output gate
어떤 정보를 output으로 내보낼지 정함
먼저 sigmoid layer에 input data를 넣어
output 정보 정한 후 cell state를 tanh
layer에 넣어 sigmoid layer의 output과
곱하여 output으로 내보냄

② forget cell
cell state에서
sigmoid layer을 거쳐
어떤 정보를 버릴것인가 결정

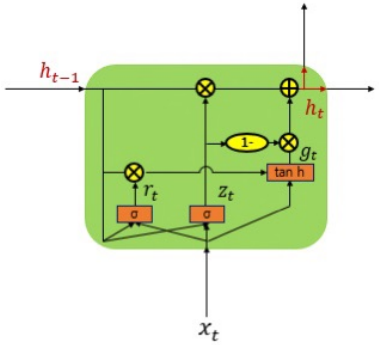
③ input gate
앞으로 들어오는 새로운 정보 중 어떤
cell state에 저장할 것인지를 정함
sigmoid layer을 거쳐 어떤값을 update 할건지 정한후
tanh layer에서 새로운 후보 vector 생성

GRU

– 파라미터가 많아지는데 데이터가 충분하지 않은 경우 오버피팅이 발생하는데 이런 단점 극복하기 위해 LSTM 변형시킨 GRU등장

LSTM 에서는 출력, 입력, 삭제 게이트라는 3개의 게이트가 존재

GRU 에서는 업데이트, 리셋 게이트 2개의 게이트만 존재 (forget, input gate를 하나의 update gate로, cell state 와 hidden state 통합)



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

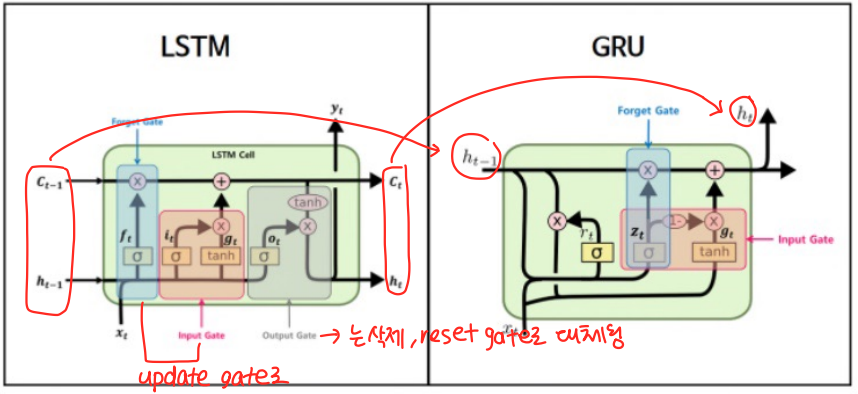
기존의 LSTM 을 사용하면서 최적의 파라미터를 찾아낸 상황에서는 굳이 GRU로 바꿔서 사용할필요 X

데이터 양이 적을때는 매개변수의 양이 적은 GRU를

〃 많을때는 LSTM

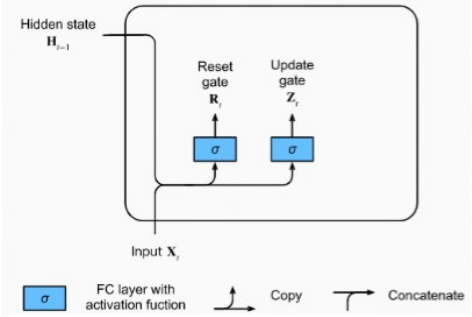
LSTM에서는 forget과 input이 독립적이었지만 GRU에서는 전체양이 정해져있어 (=1), forget한 만큼 input하는 방식으로 제어. 이는 gate controller인 z(t)에 의해 조절된다

z(t)=1 → forget gate 열리고, 0이면 input gate 열림



update gate로

output gate는 삭제됨

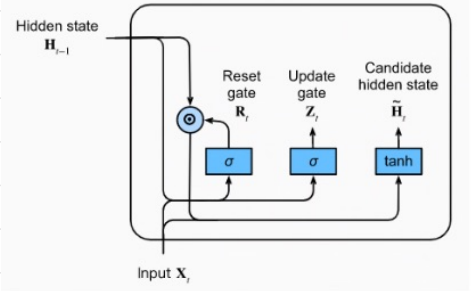


[1] Reset gate, r(t)

: 이전의 정보를 얼마나 잊어야하는지를 결정합니다.

[2] Update gate, z(t)

: 이전의 정보(기억)을 얼마나 통과(유지)시킬지를 결정합니다.

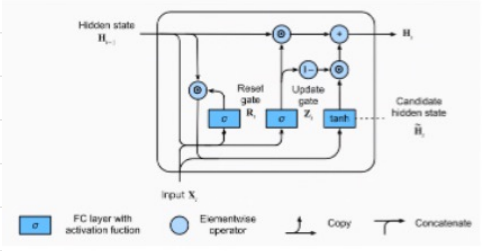


: hidden layer의 후보(candidate)로 reset gate인 r(t)와 Uh(t-1)을 곱하여, 이전 타임 스텝에서 무엇을 지워야할지를 결정합니다.

: tanh 함수는 -1 에서 1 사이의 값을 가지므로, candidate hidden state도 -1에서 1사이 값을 가집니다.

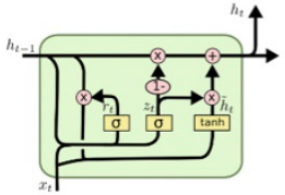
: updat gate를 아직 반영하지 않았기 때문에 아직까지는
'candidate' hidden state입니다.

: reset gate 를 모두 1, update gate를 모두 0으로 설정한다면, 모델은 Vanilla RNN과 같아집니다.



: 이제 update gate를 합칠 차례입니다. update gate는 얼마나 새로운 hidden state(h(t)) 가 이전 hidden state(h(t-1)과 같을지 아니면 새로운 cadidate hidden state이 많이 사용될지를 결정합니다.

: z(t)가 0이라면 old state(h(t-1))을 유지하고, 1이라면 candidate state(tilda h(t))로 완전히 대체됩니다.



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

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

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

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

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

이전 시점의 hidden state 와 현시점의 값을 활성화함수 시그모이드를 적용하여 구하는 방식, 0~1 사이의 값을 가지고 이전 hidden state 의 값을 얼마나 활용한것인가에 대한 정보

v update gate : LSTM의 input, forget gate와 비슷한 역할을 하며 과거와 현재의 정보를 얼마나 반영할지에 대한 비율을 구하는것이 핵심

(1) 식을 통해서 구한 결과 z는 현재 정보를 얼마나 사용할지 반영

(1-z)는 과거 정보에 대해 얼마나 사용할지 반영,

최종적으로 (4)식을 통해 현 시점의 출력값 hidden state 구할 수 있음

summary : ① reset gate는 short-term dependency 의미

② update gate는 long-term dependency 의미

(feature , label , window_size=20)

feature = [1,2,3, ··· , 40] len(feature)=40

label = [1,2,3, ··· 40] len(label)=40

feature_list=[ ], label_list=[ ]

for i in range(40−20) 20번 반복

feature_list.append(feature[i : i+window_size]) → 20개씩 자르는 역할

label_list.append(label[i+window_size]) → 그 다음 정답 저장해놓겠다

<label>

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40

(10 is written beside the row that starts with 11.)

<feature>

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40

i= 0
1
2
3
4
⋮

label [0+20] label[20]

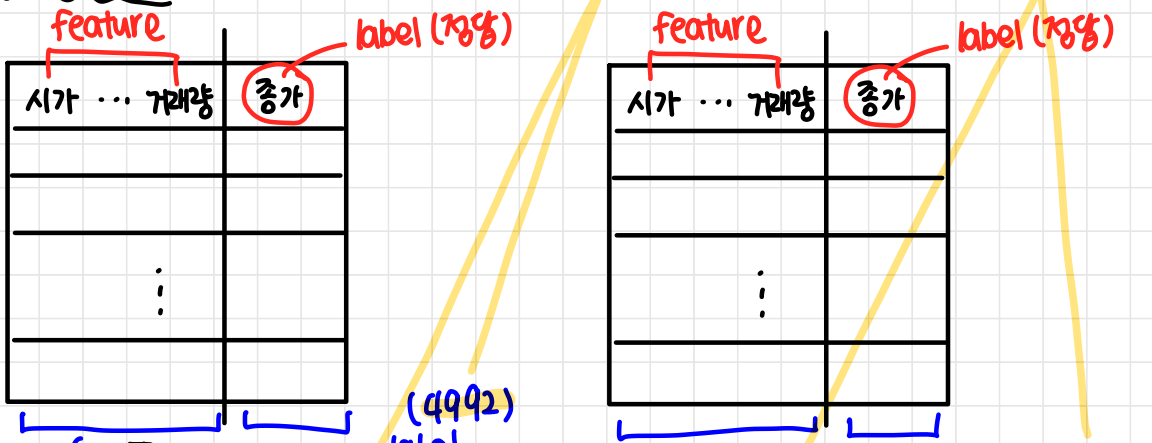
[1+20]

⋮

label [19+20]

scaled_df (정규화시켜준 데이터)

전체데이터 : 5212 / Train 데이터 : 5012 / Test 데이터 : 200



⇒ 20개씩 차이가 나는 이유는 데이터셋 정의해주는 함수 make_dataset 때문에 (window_size=20 으로 정해줬기때문)

x_train (3494, 20, 4)

x_test (1498, 20, 4)

- batch size (data 개수)
- window_size
- feature 개수