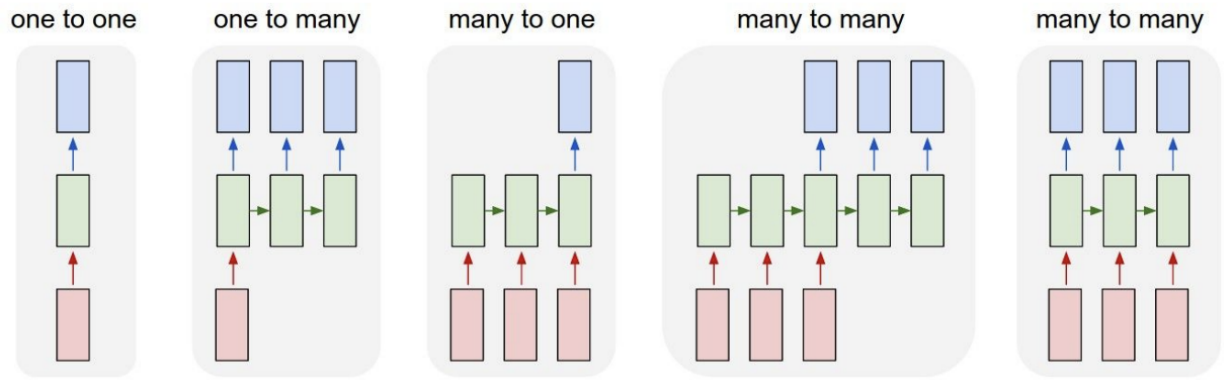
Lecture10_Recurrent Neural Networks

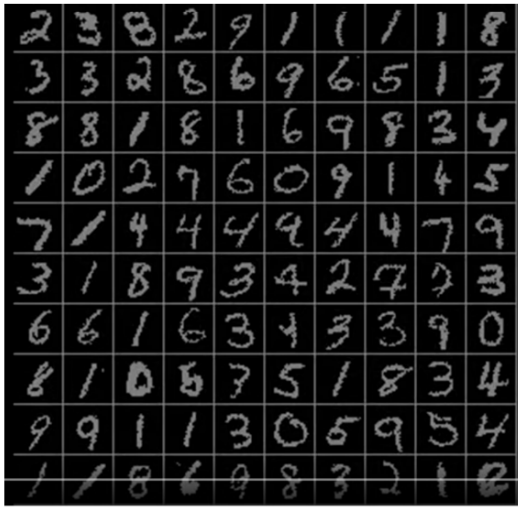


one-to-one	one-to-many	many-to-one	many-to-many	many-to-many
(vanilla neural network) **CNN**	image captioning — 사진(fixed size data)으로부터 문장(sequence of word) 만들어냄	Sentiment classification: seq of words → sentiment	Machine Translation: seq of words → seq of words	frame-level video classification — 매 프레임에서 classification

→ RNN은 이러한 가변 길이를 가지는 데이터에 강함

Sequential Processing of Non-Sequence Data

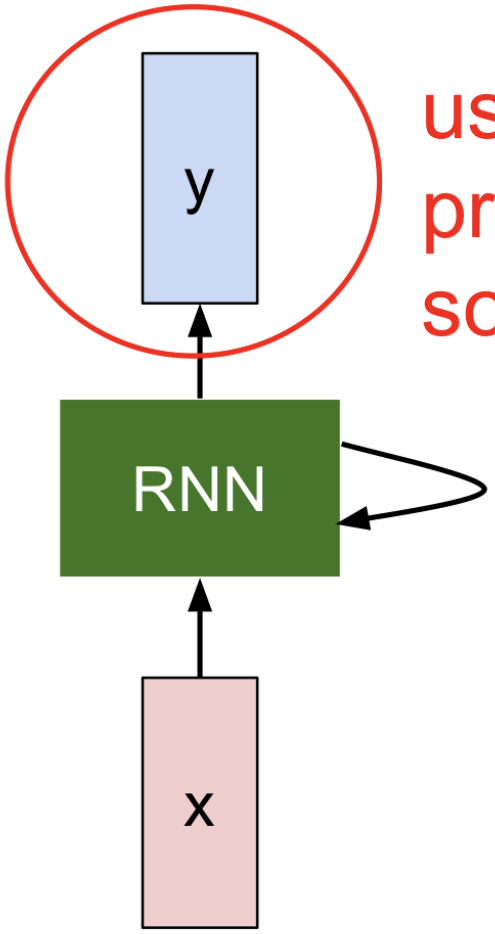
Classify images by taking a series of “glimpses”



Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with

하지만 이렇게 입출력은 고정이지만, sequential processing이 필요한 경우에도 유용하다.

예) 이미지(고정 길이 입력)에서 숫자 분류를 할 때, feed forward pass 한 번이 아니라, 이미지의 여러 부분을 조금씩 살펴본다.



usually want to predict a vector at some time steps

function form

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

new state

some function with parameters W

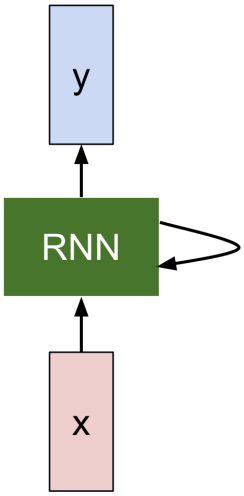
old state

input vector at some time step

이전 hidden state에서의 output과 현재 step의 input vector(x_t)가 입력으로 들어옴. hidden state는 입력값 마다 매번 update됨(hidden state를 담당하는 가중치가 따로 있음)

⇒ 같은 function과 parameter들이 매 스텝마다 업데이트됨!

Vanilla Recurrent Neural Network

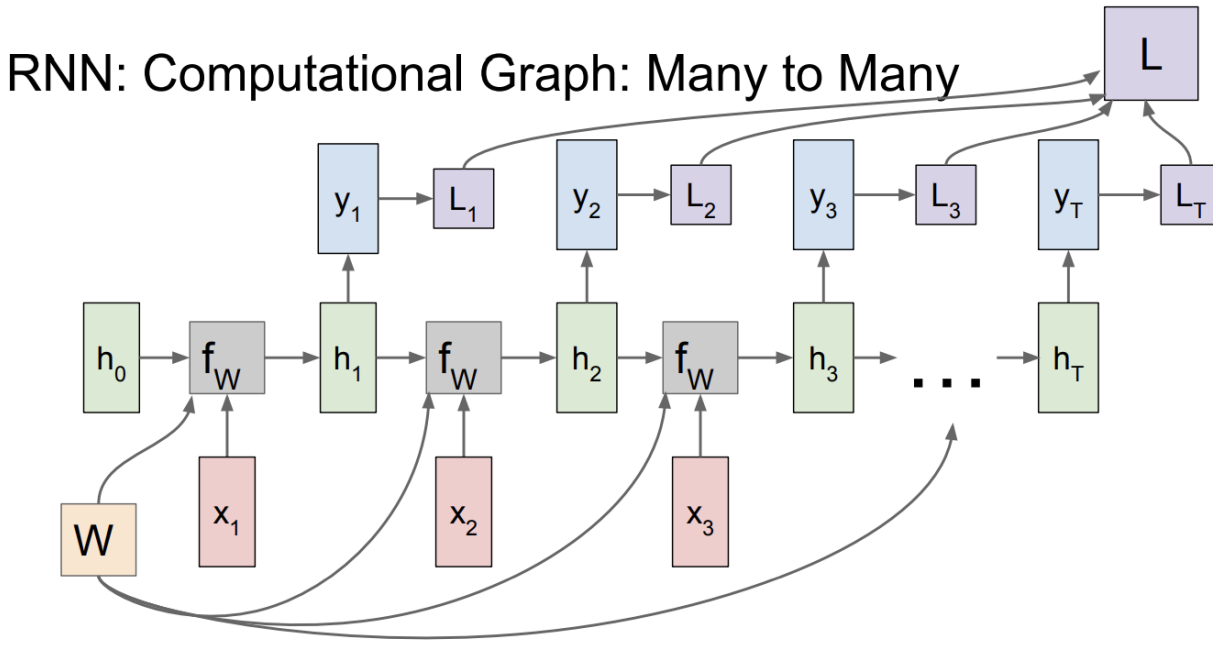


$$h_t = f_W(h_{t-1}, x_t)$$

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

$$y_t = W_{hy}h_t$$

RNN — Computational Graph: Many to Many



*. h0은 일반적으로 0으로 초기화

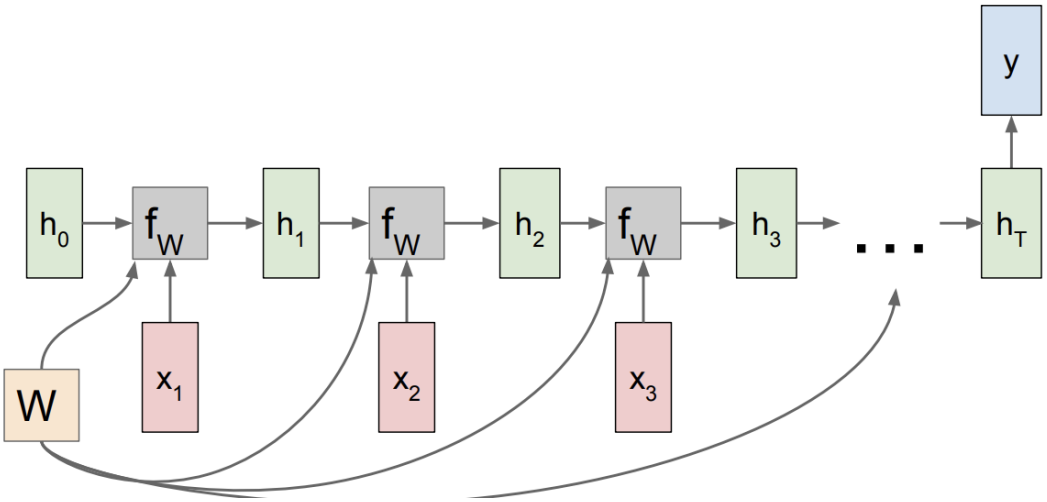
계속해서 똑같은 가중치 W를 re-use. (h와 x는 달라지지만 W는 같음)

backpropagation시에, 각 스텝에서의 W에 대한 그레디언트를 모두 계산해서 더해줌.

RNN의 출력값 h_t가 또 다른 네트워크의 입력으로 들어가서 y_t(예를들어 매 스텝의 class score)를 만들어 냄 → 만일 각 시퀀스마다 ground truth가 있다면 각 스텝마다 개별적으로 y_t에대한 loss(e.g. softmax loss)가 계산 가능 함 ⇒ 최종 loss: 그러한 loss들의 합

RNN — Computational Graph: Many to One

RNN: Computational Graph: Many to One

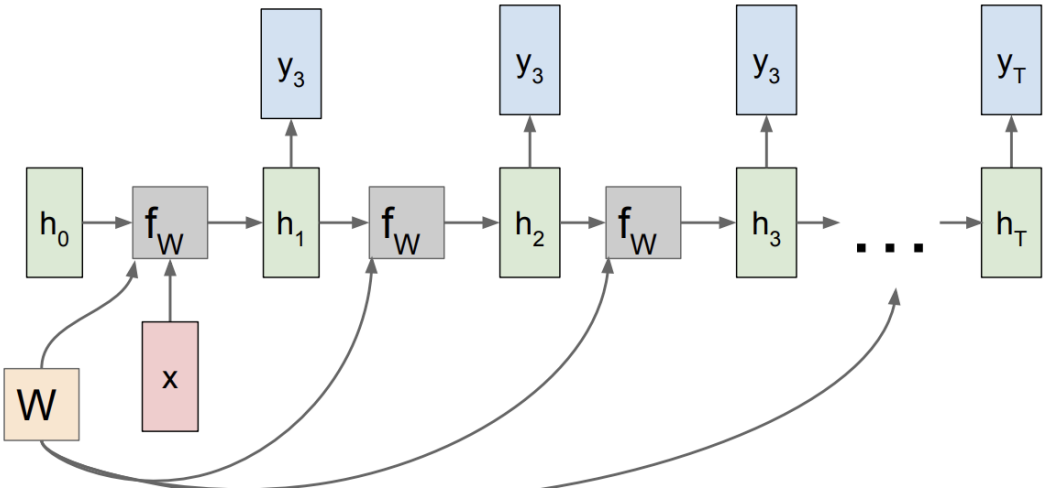


- e.g. 감성분석
- final hidden state가 전체 sequence에서 모든 context에대한 요약임

  → final state에대한 결과값만 나옴

RNN — Computational Graph:One to Many

RNN: Computational Graph: One to Many

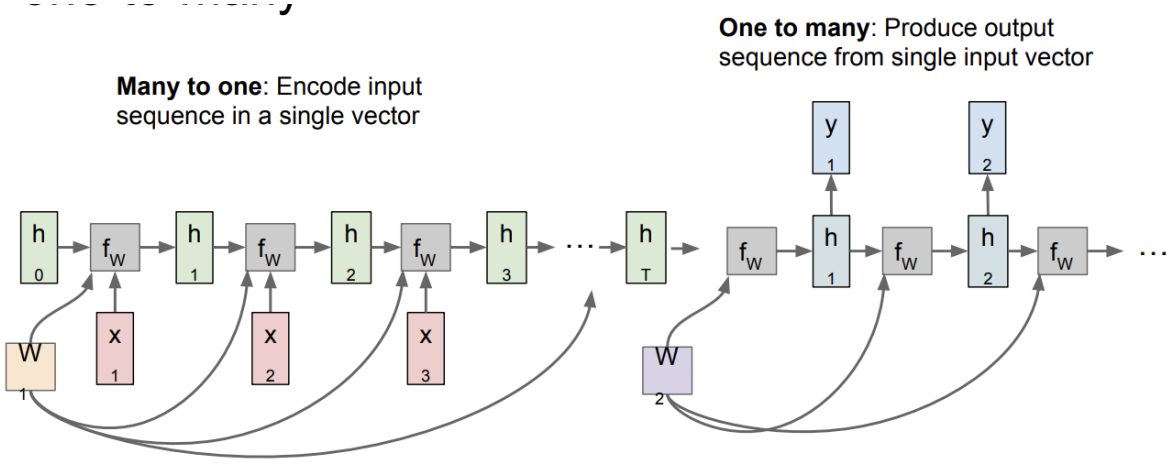


- fixed input과 가변 출력을 가지는 경우

*. 대부분 고정길의 입력으로 initial hidden state를 초기화함

Sequence to Sequence: Many-to-one + one-to-many

: encoder-decoder 구조



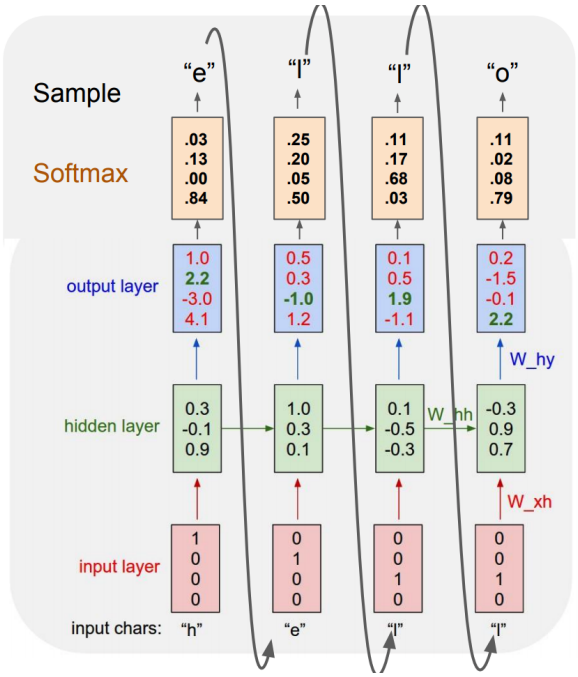
encoder가 가변 입력을 받아 정보를 요약한 한개의 vector로 출력하고, 이를 다시 디코더의 입력으로 넣어 가변 출력을 내놓음.

예) 프랑스어 문장 → 문장이 요약된 single vector(context vector) → 프랑스어 문장

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

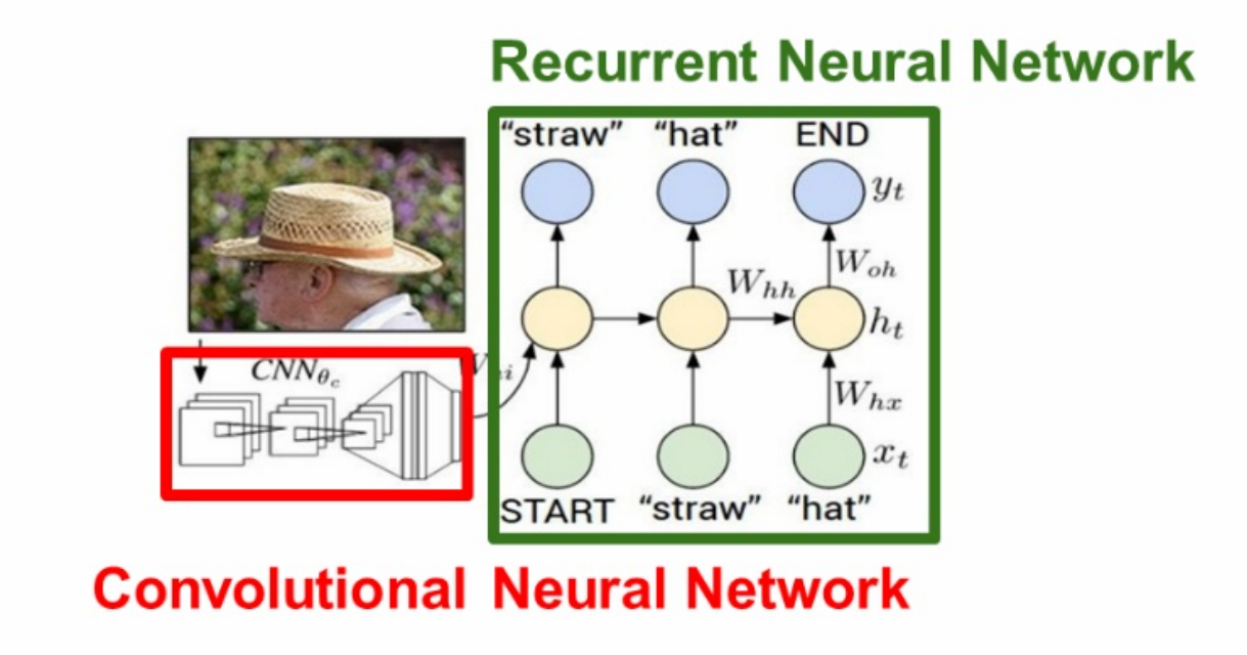


-모든 timestep에서 다음에 올 단어를 예측하고 오차를 크로스 엔트로피를 통해 구한다.

-RNN은 매우 느리고(순착적으로 진행해야하기때문) 많은 메모리를 사용하는 단점이 있음

→ 이를 해결하기 위해 Truncated Backpropagation through time을 사용함.(CNN의 미니배치 같은 것)

Image Captioning



CNN과 RNN을 결합한 기술. CNN으로 사진을 넣으면 RNN에서 단어 seq가 출력으로 나온다.

: 이미지와함께 <start> 시그널을 함께 넣어주고, 소프트맥스 직전 FC 레이어에서 이미지가 요약된 벡터를 RNN으로 보내준다. 그리고 RNN의 hidden state를 하나씩 거쳐 captioning된 단어를 하나씩 출력하다 <END> 토큰을 만나면 멈춘다.

with attention개념도 있는데, 이 부분은 나중에 좀 더 자세히 정리..

LSTM

RNN의 GVP 해결 위해 등장

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

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

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

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

- **Cell** remembers values over arbitary time intervals (**Memory**): cell은 임의의 시간동안 값을 기억하여 memory 역할을 한다.

  short term 메모리는 hidden state에 있음.

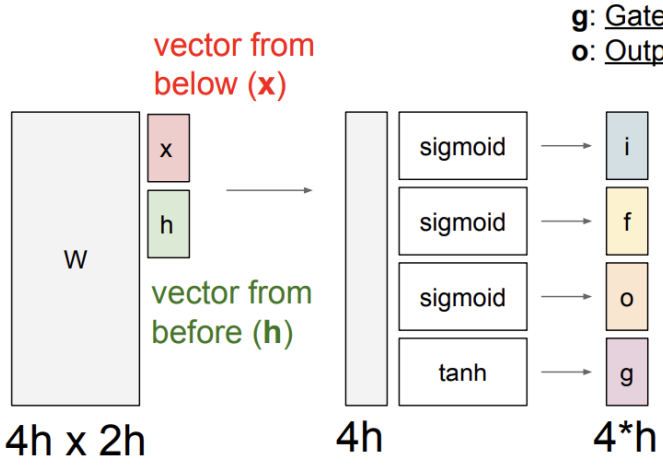
  LSTM은 이에 cell이 long-term memory 역할을 함. (Ct가 memory가 되어 time step마다 전달됨)

- gate들로 logn term이 short term에 얼마나 반영될지 등을 조절함

_ong Short Term Memory (LSTM)

'Hochreiter et al., 1997]

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



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

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

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

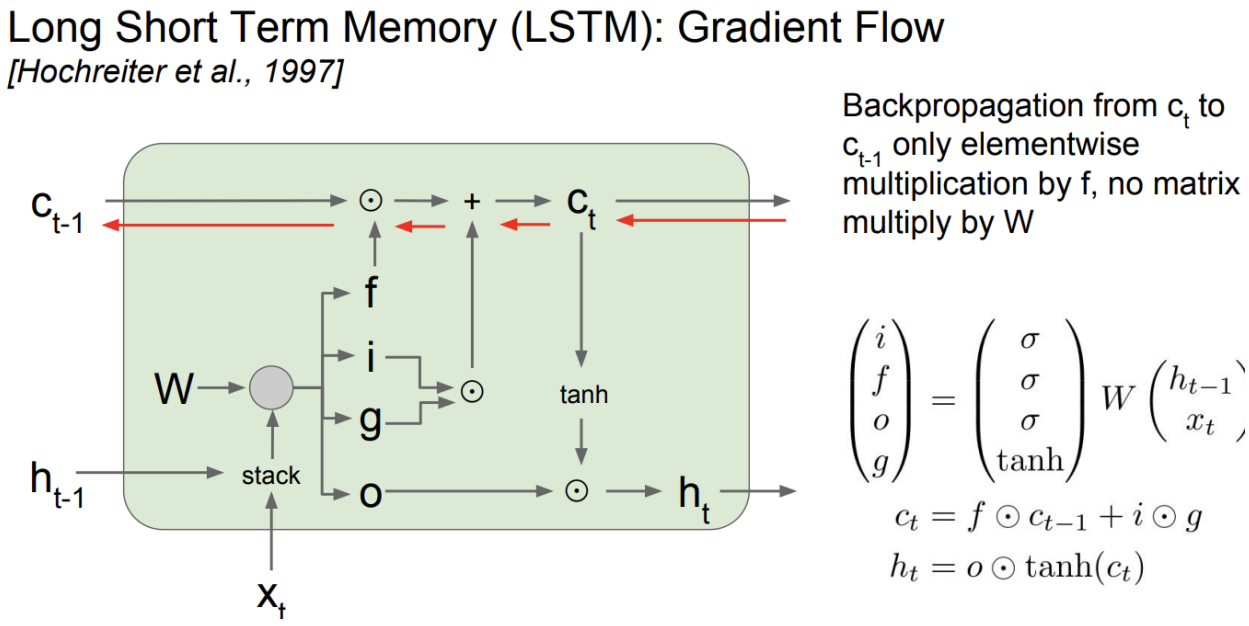
- **Forget gate** f_t: whether to erase cell. cell에 저장되어있는 기억을 잊을것인지를 결정한다.

즉, 1이면 Ct-1이 반영되어 정보를 보존하는 것이고, 0이면 Ct-1이 반영되지 않는다.

- **Input gate** i_t: whether to write cell.input을 쓸 것인지를 결정한다.
- **Output gate** o_t: how much to reveal cell. 현재 cell이 hidden state에 반영이 될 것인지를 결정한다.

- 과거 state인 Ct-1를 업데이트해서 새로운 cell state인 Ct를 만든다. 우선 이전 state에 ft를 곱해 잊어버리기로 했던 것을 잊어버리고, it * Ct_hat을 더해준다.
- 그리고 output을 낸다. sigmoid layer에 input을 태운 ot로 어떤 부분을 아웃풋으로 내보낼지 정한다. 이를 cell state를 tanh layer에 태운 값과 곱해주면 원하는 부분만 output으로 내보낼 수 있는 것이다.

LSTM gradient flow



LSTM은 RNN의 gradient vanishing problem을 해결하는가?

— LSTM의 cell state를 보면, i_t와 o_t는 덧셈이 되기때문에 소수값에 영향을 받지않고 그대로 더해진다. 이에 0~1 소수값을 가지는 f_t가 곱해져 GVP가 일어날 수 있지만 훈련으로 f를 1에 가깝게 만들 수 있다.

— 만일 f가 1에 가깝다면 GVP가 줄어들 수 있다. 이는 곧 f의 의미에 의해 이전 state의 cell을 거의 반영한다는 뜻으므로 긴문장에서도 앞에 있는 단어를 잊지 않을 수 있는 것이다.