

Week12_CS231n Recurrent Neural Networks

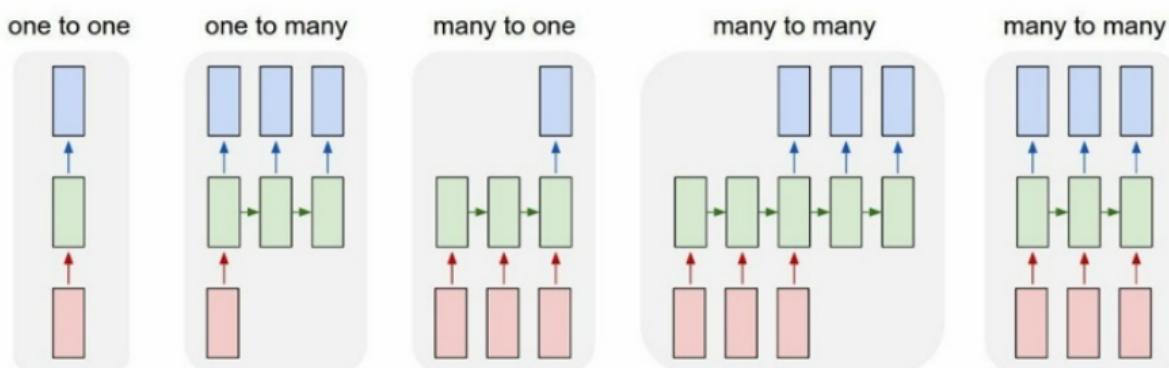
1. Recurrent Neural Networks : Process Sequence

이전에 배운 아키텍쳐들은 “one to one” 방식으로 하나의 입력이 하나의 출력은 내보낸다.

Machine learning에서는 모델이 다양한 입력을 처리할 수 있어야한다.

⇒ RNN network

Recurrent Neural Networks: Process Sequences



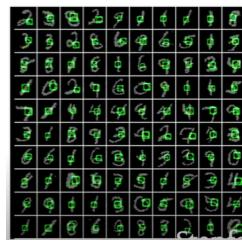
- One to many (ex. Image Captioning) :
 - Input : fixed (ex. image)
 - Output : sequence of variable length (ex. caption)
- Many to one (ex. Sentiment Classification) :
 - Input : variably sized (ex. piece of text), (ex2. video with many frames)
 - Output : sentiment of text (ex. positive / negative), (ex2. classification)
- (1) Many to Many (ex. Machine Translation)
 - Input : (ex. English sentence)
 - Output : (ex. French sentence)
 - 두 언어 문장의 길이가 다르므로 variable input/output이 가능한 모델
- (2) Many to Many

- (ex. Video classification on frame level)

⇒ Recurrent Neural Networks 는 variable length의 data를 다루기 위한 일반적인 paradigm이다.

2. Sequential Processing of Non-Sequence Data

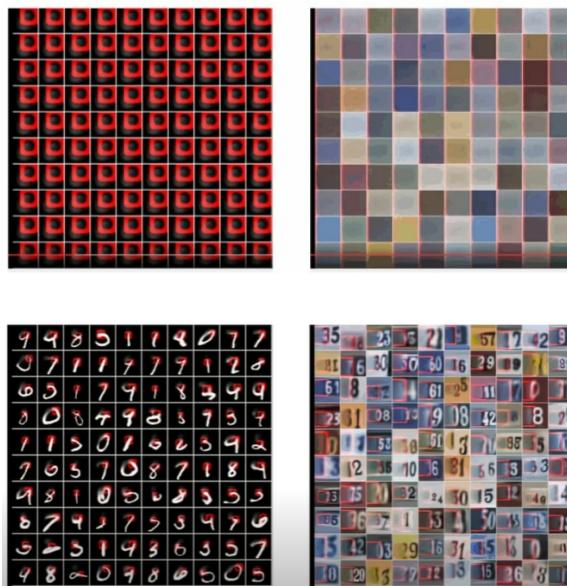
RNN은 입/출력이 fixed된 상황에서도 유용하다.



이미지의 숫자를 분류하는 문제에서 input image의 정답을 feed forward pass 한번으로 결정하는 것이 아니라 이 네트워크는 이미지의 여러 부분을 조금씩 살펴본다.

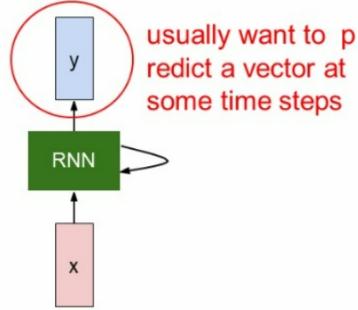
<Generate images>

동일한 방법으로 이미지를 생성할 수 있다. 순차적으로 전체 출력의 일부분을 생성해나간다.
(Output : fixed)



3. Recurrent Neural Network

(RNN의 동작 방식)



RNN은 위 그림의 Recurrent Core Cell을 가지고 있음.

Input x 가 RNN에 들어가면 RNN 내부의 hidden state가 input이 들어올 때마다 업데이트 된다.

hidden state는 모델에 feedback되고 새로운 input x 가 들어온다.

출력값을 내보낸다.

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

new state old state input vector at
 some function some time step
 with parameters W

수식으로 살펴보면 위와 같다.

f : parameter W 를 가진 함수,

이전 상태인 hidden state h_{t-1} 과 현재 상태 입력인 x_t 를 입력으로 받는다.

$\Rightarrow h_t$ 를 출력하고 h_t 는 다음 상태의 hidden state가 된다.

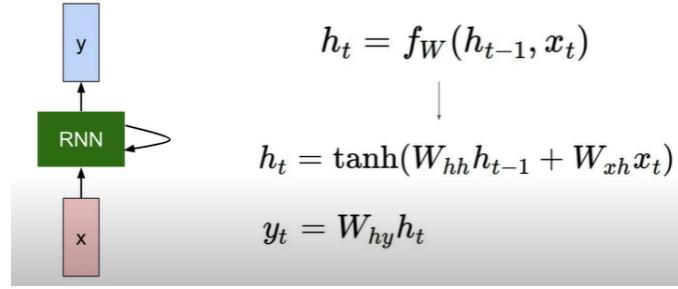
Output y 를 가지려면 h_t 를 Input으로 같은 FC-layer를 추가해야 한다.

Notice : 함수 f , 파라미터 W 는 매 step 동일하다.

(The same function and the same set of parameters are used at every time step.)

Vanilla Recurrent Neural Network

The state consists of a single “hidden” vector h



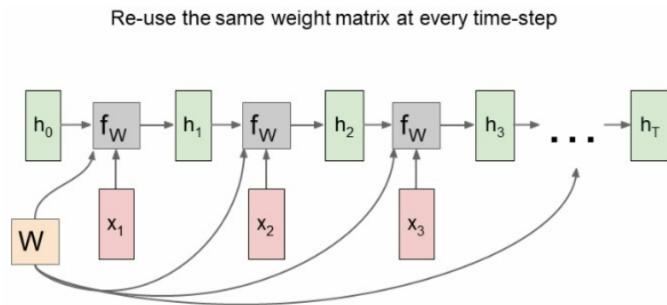
W_{hh} : hidden state에서 오는 W , 이전 hidden state인 h_{t-1} 과 곱해지는 값

W_{xh} : x 에서 RNN으로 들어오는 weight, 현재 입력 x_t 과 곱해지는 값

W_{hy} : RNN cell에서 y 로 넘어가는 weight

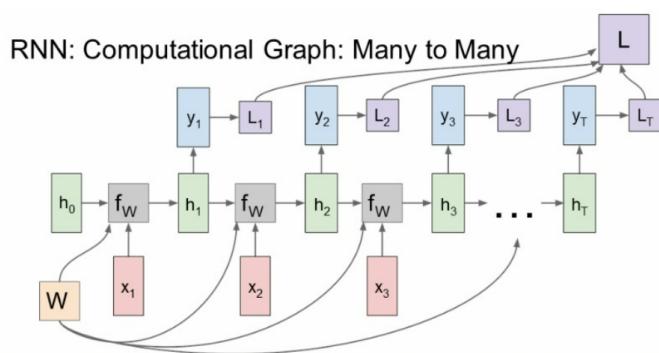
tanh : activation function

RNN : Computational Graph



⇒ 동일한 가중치 행렬 W 가 사용된다.

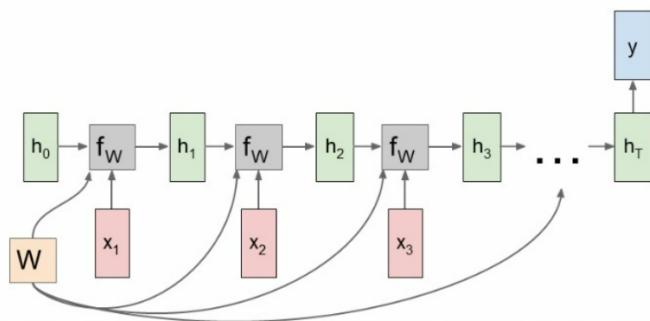
Many to Many



- W gradient :

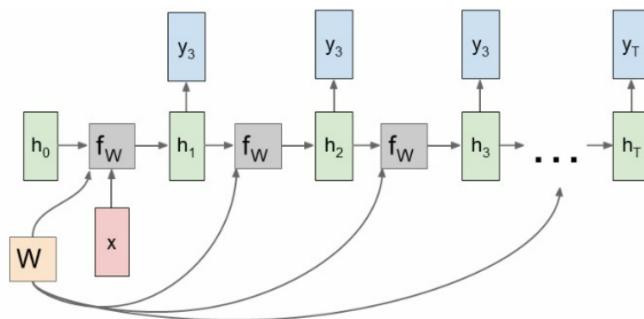
- Backprop에서 W 의 그레디언트를 구하려면 각 스텝에서의 W 에 대한 그레디언트를 전부 계산한 뒤 이 값을 더하면 된다.
- Loss :
 - 각 시퀀스마다 Ground truth label이 있는 경우 각 스텝마다 y_t 에 대한 Loss를 계산할 수 있다. \Rightarrow 전체 loss의 합 = 최종 Loss
- $d\text{Loss}/dW$
 - 각 step마다 가중치 W 에 대한 local gradient를 구할 수 있음 \Rightarrow 개별로 계산된 local gradient를 최종 gradient에 더한다.

Many to one



최종 hidden state에서만 결과값이 나온다.

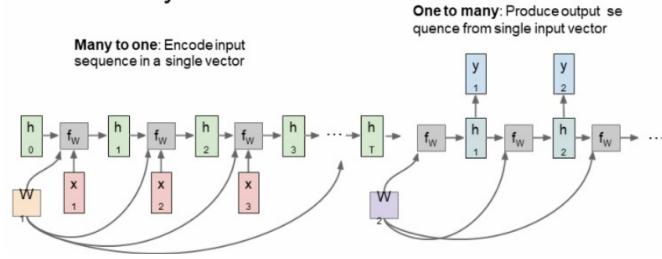
One to Many



고정입력은 모델의 initial hidden state를 초기화시키는 용도로 사용한다.

Sequence to Sequence : Many to one + one to many

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



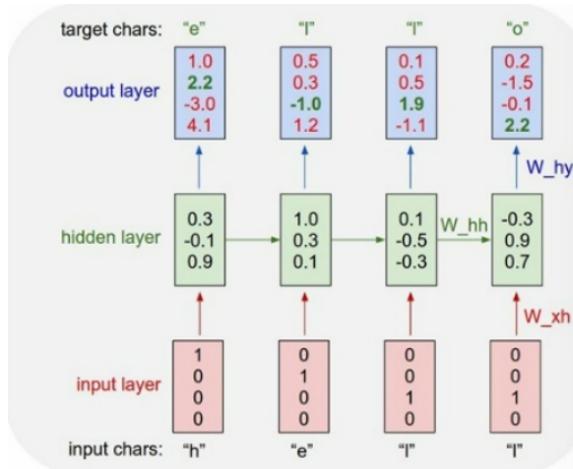
번역기같은 곳에서 사용.

Variable length input과 Variable length output

Encoder : Variable length input을 받는다. \Rightarrow 전체 sentance 요약

\Rightarrow Decoder : Variable length output

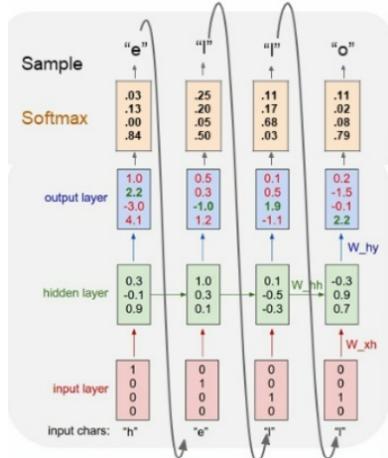
ex) Character-level Language Model



Input : h, e, l, l \Rightarrow 다음 문자 예측

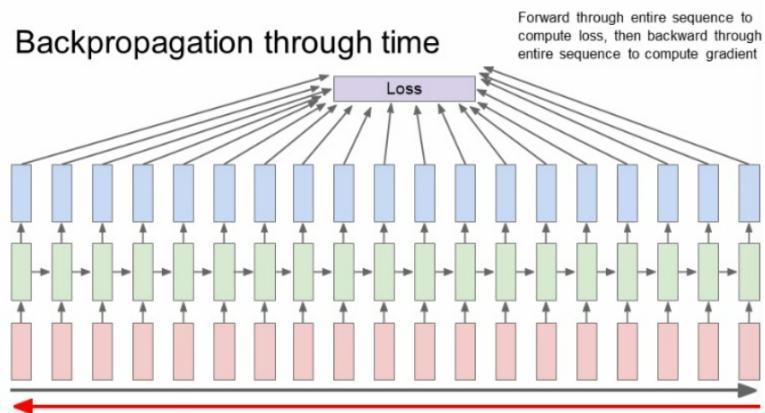
Output : e, l, l, o

- Train time : Training sequence로 hello각 단어를 넣어준다.
 - one-hot encoding 사용
 - 반복훈련



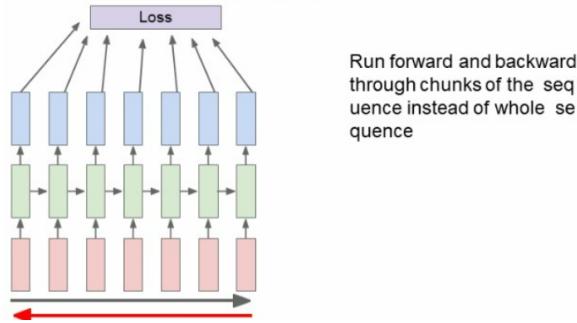
- Test time : Input값이 들어가면 단어 모델을 만들어 Output이 나오고 이것이 다음 Input값이 됨
 - Score를 확률 분포로 표현하기 위해 softmax 함수 사용

Backporpagation through time



- Forward pass : 전체 sequence가 끝날 때까지 출력값 생성
- Backward pass : 전체 sequence가 끝날 때까지 출력값 생성 \Rightarrow 시퀀스가 길 경우 gradient 1회 계산까지 너무 오래 걸림, 비효율적

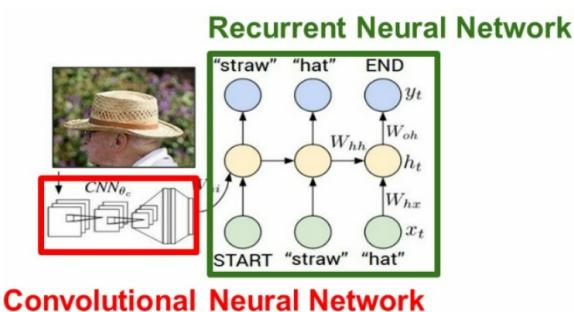
Truncated Backpropagation



실제로는 truncated backpropagation을 사용한다.

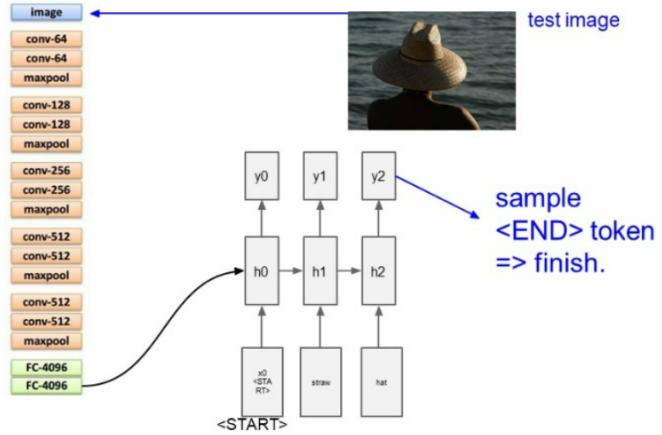
- train time에 한 스텝을 일정 단위로 자른다.
 - 100개를 잘랐다고 하면 100개를 forward하고 여기서 loss를 계산
 - Gradient step 진행
 - 반복
- 다음 hidden state는 이전 hidden state를 사용한다.

Image Captioning



CNN과 RNN을 혼합한 대표적인 방법, 입력은 이미지가 들어가지만 출력은 자연어로 된 caption이 나온다.

CNN이 이미지 정보 vector를 출력하면 RNN초기 step으로 들어가 RNN이 문장을 만들어 낸다.



Test에서 마지막의 softmax score를 사용하지 않고 직전의 fc-4096 vector를 출력으로 한다
⇒ 전체 이미지 정보를 요약

RNN에는 초기값으로 <START> 신호를 준다.

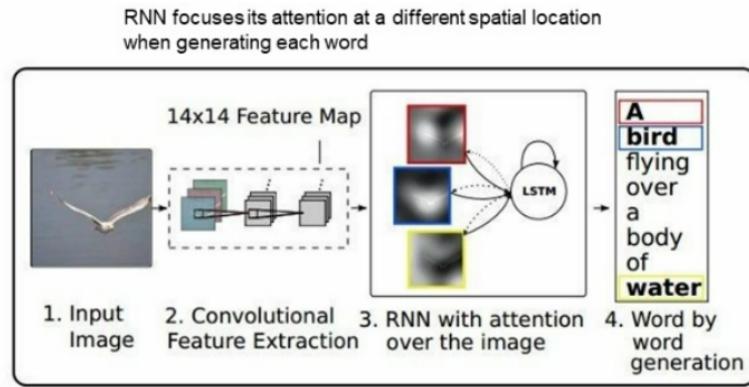
반복적으로 sampling 후 끝나면 <END>이 나온다.

(Train time에는 모든 caption의 종료지점에 <END> 토큰을 삽입) ⇒ 토큰 넣는 것도 학습

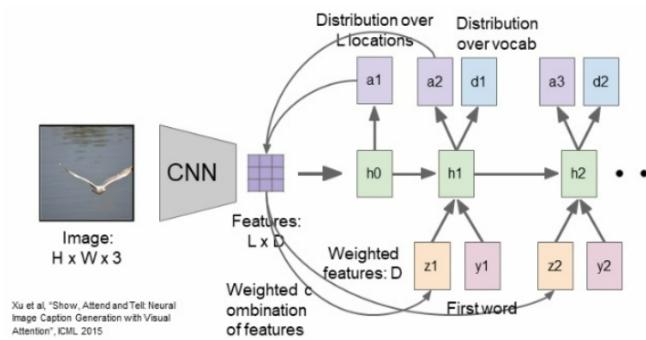


Image를 학습시키기 위해서는 natural language caption이 있는 이미지가 있어야한다. (대표적인 Microsoft COCO)

Image Captioning with Attention



Caption을 생성할 때 이미지의 다양한 부분을 attention해서 볼 수 있다.

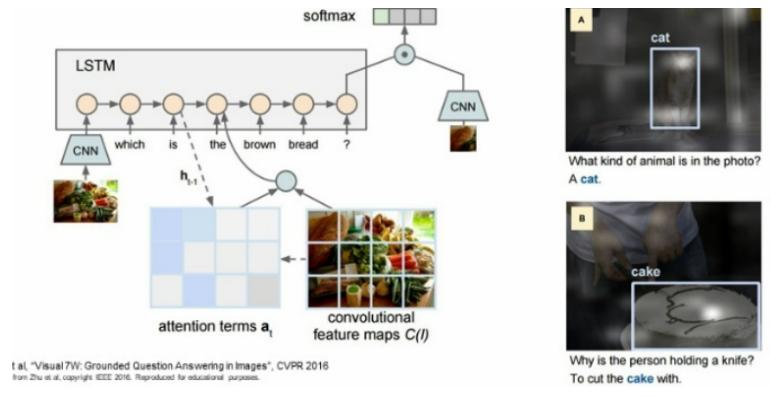


Visual Question Answering

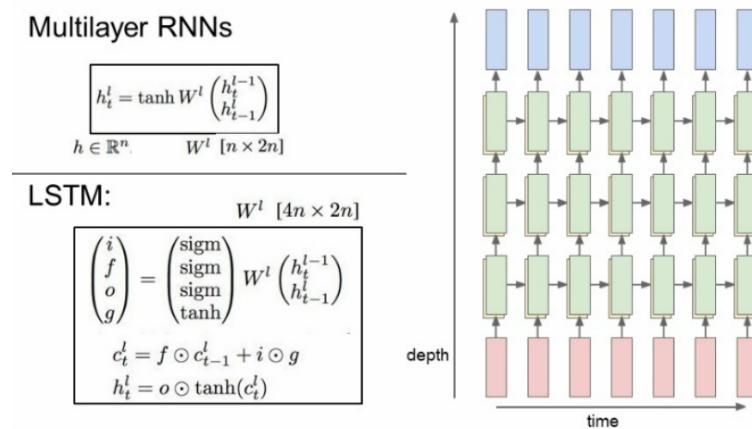


Input : 2개 (이미지, 이미지와 관련된 질문)

⇒ RNN으로 만들 수 있다.

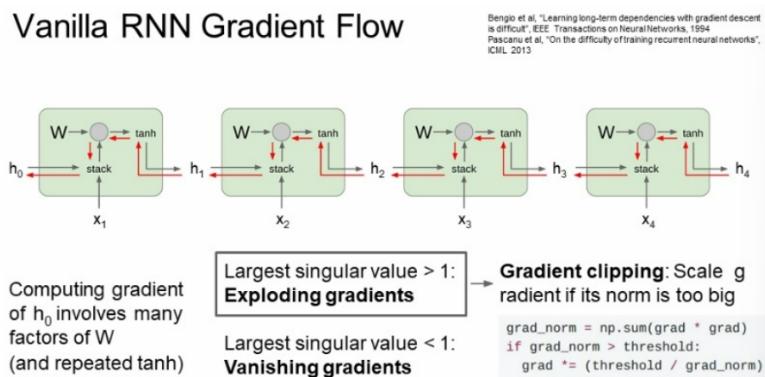


Multilayer RNNs



보통의 RNN은 multilayer로 사용한다. 모델이 깊어질수록 성능이 좋아지기 때문이다.

Vanilla RNN Gradient Flow



보통 Vanilla RNN은 잘 사용하지 않는다. RNN을 학습시킬 때 Gradient 문제가 발생하기 때문이다.

Backprop 단계에서 매번 RNN cells를 통과할 때마다 가중치 행렬의 일부를 곱해야 한다. \Rightarrow 가중치 행렬들이 개입하게 되고 비효율적

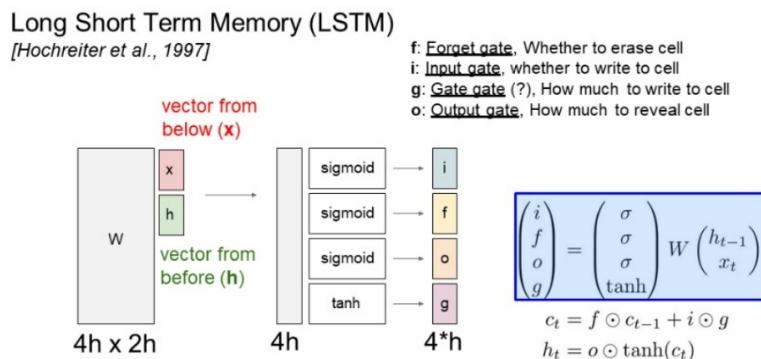
\Rightarrow 매번 곱해지는데 곱해지는 값이 1보다 크면 exploding, 1보다 작으면 vanishing 문제가 발생

\Rightarrow Gradient exploding은 Gradient clipping으로 막을 수 있지만 Gradient Vanishing은 막지 못한다.

Long Short Term Memory(LSTM)

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 state라는 두 번째 벡터 존재
- H_{t-1}, x_t 를 입력으로 받는다.
- ifog gate를 계산한다.
- c_t 를 업데이트한다 \Rightarrow 다음 스텝의 hidden state 업데이트



i : input gate : cell의 입력 x_t 에 대한 가중치 (현재 정보를 기억하기 위한)

f : forget gate : 이전 스텝의 cell을 얼마나 잊을지 결정 (과거 정보를 잊기 위한)

o : output gate : cell state를 얼마나 밖에 드러낼것인지 (현재 정보를 기억하기 위한)

g : gate gate : input cell을 얼마나 포함시킬지