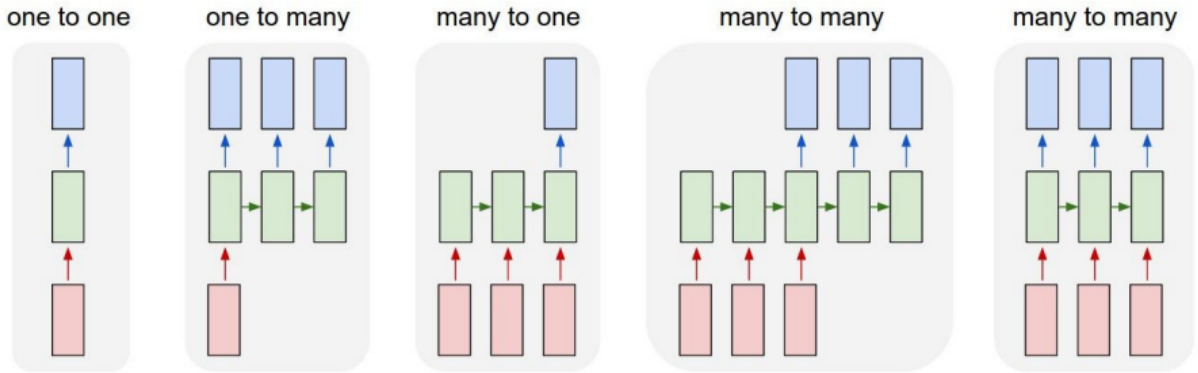
RNN(Recurrent Neural Network)

Vanilla Neural Network(이전까지 배운 구조)의 한계는 고정된 크기의 vector를 입력으로 받아들이고 고정 크기의 vector를 출력. hidden layer를 걸쳐 고정된 양의 계산 단계를 사용하여 mapping 수행.

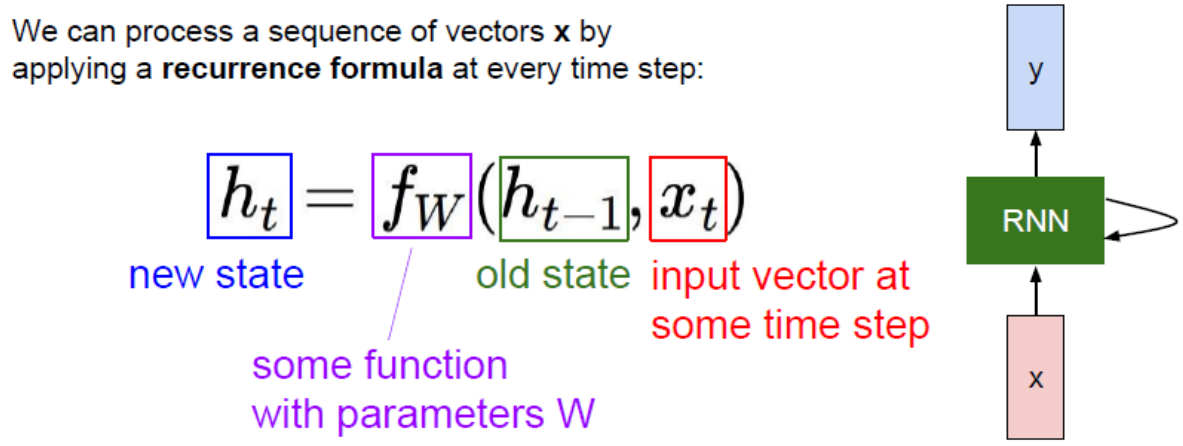
RNN은 vector sequence에 대해 연산을 수행할 수 있고, 구체적으로 만들 수 있음. 다양한 입 / 출력을 다룰 수 있도록 만든 모델.



빨간색 box : input vector, 파란색 box : output vector, 녹색 box: RNN state

화살표 : function

1. one to one : RNN이 없는 Vanilla mode.
2. one to many : Sequence output
3. many to one : Sequence Unit
4. many to many : Sequence input and sequence output
5. many to many : Synced sequence input and output



일부분씩 순차적으로 처리할 수 있으므로 입출력이 고정된 길이라고 해도 "가변 과정(processing)"인 경우에 RNN은 유용

일반적으로 RNN은 작은 *Recurrent Core Cell*을 가짐.

입력 x가 RNN으로 들어가면, RNN은 새로운 state vector를 만들어내기 위해 fixed function을 사용하여 input vector와 output vector를 결합.

RNN 내부의 *hidden state*에서 새로운 입력을 받아들여 update 되는 방식.

이런 방식으로 RNN은 네트워크가 다양한 입출력을 다룰 수 있는 여지를 제공

RNN 동작

1. Sequence vector xt를 받음.
2. *hidden state*를 업데이트.
3. 출력 값을 내보냄.

RNN에서 출력 값을 가지려면 ht를 입력으로 하는 FC-Layer를 추가. FC-Layer는 매번 업데이트 되는 Hidden State(ht)를 기반으로 출력 값을 정함.

*함수 f와 parameter W는 매 step에서 동일.

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

$$\downarrow$$

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

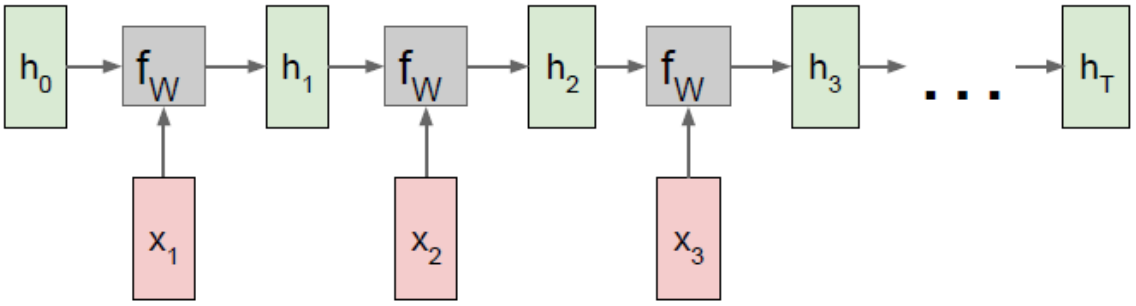
$$y_t = W_{hy}h_t$$

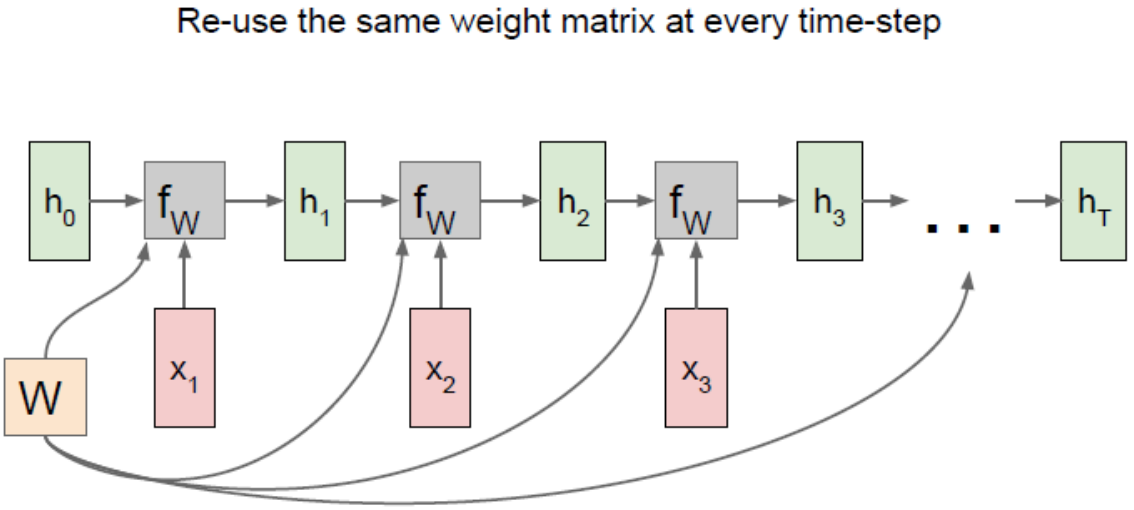
여기서 비선형성을 위해 tanh 함수를 사용.

RNN은 hidden state를 가지며 *Recurrently* feed back하는 특징

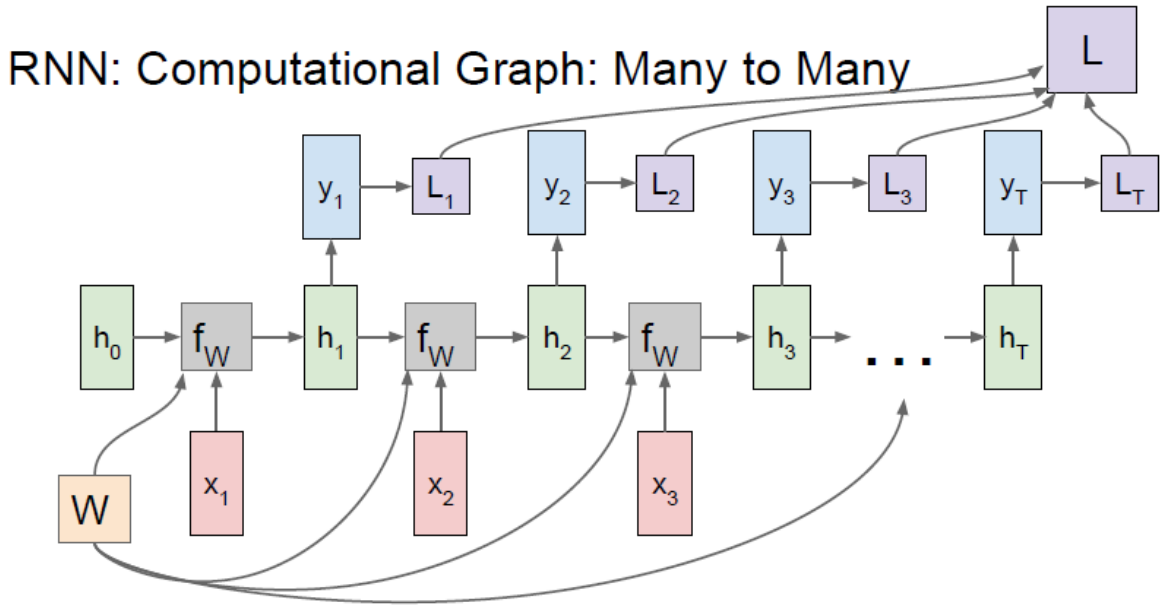
RNN의 Computational Graph

처음의 h0는 대부분 0으로 초기화. 출력인 h1가 다음에는 입력으로 들어감. 또 다른 x2를 받아 다시 출력으로 h2를 만들어 냄. 이 과정을 반복.

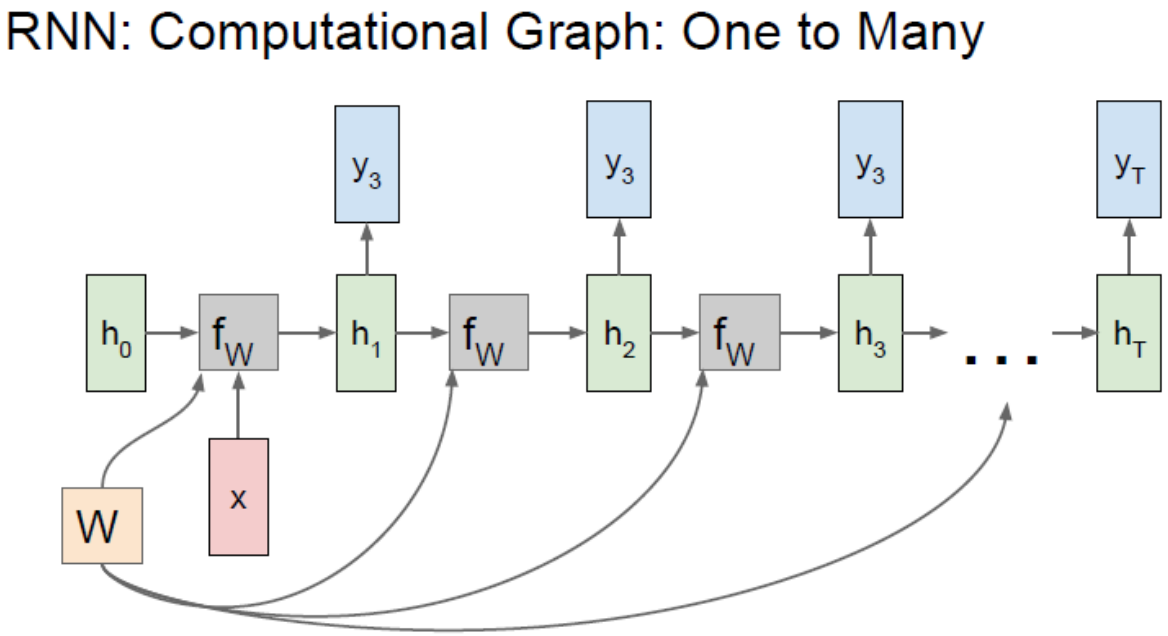




만약 출력이 다수라면 ‘many to many’ 분류

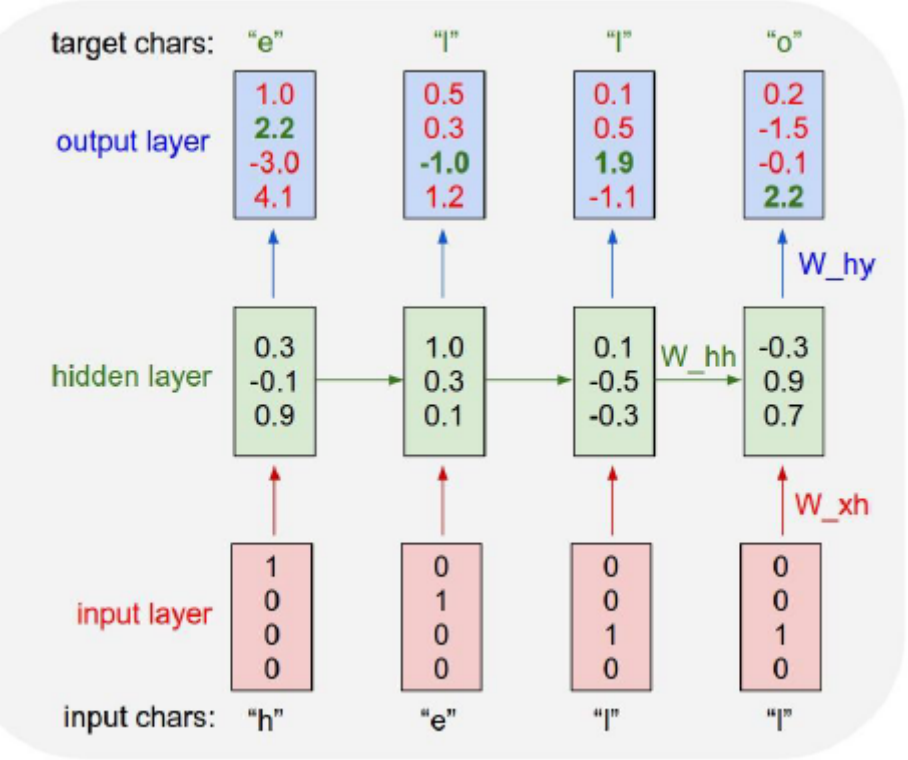


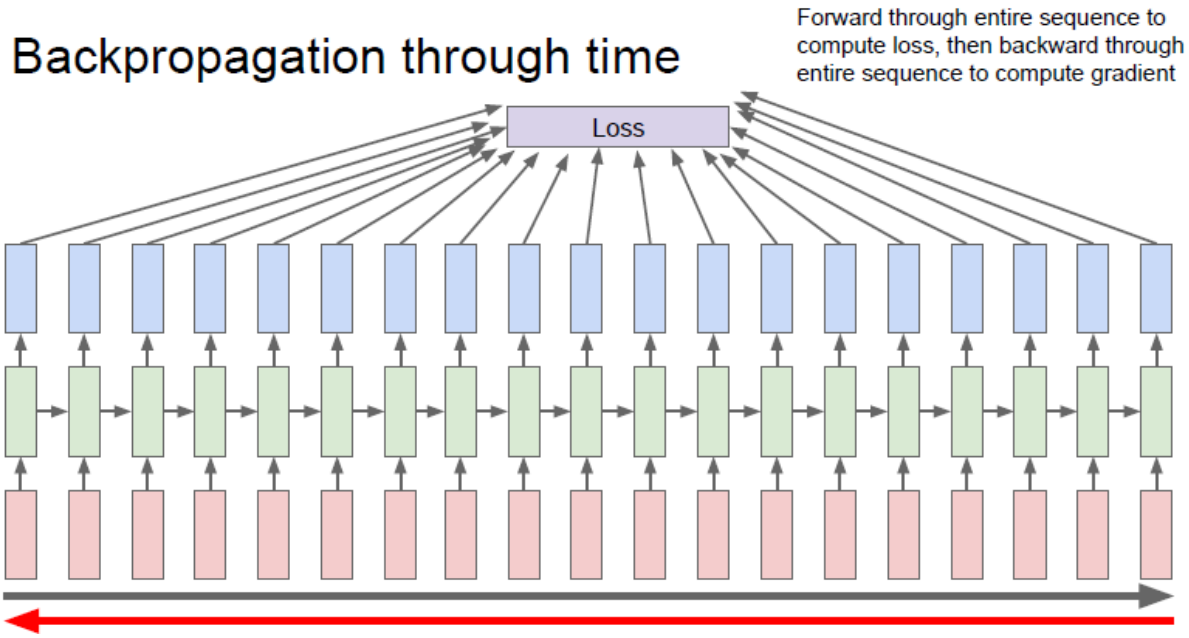
- 입력이 하나이고, 출력이 다수



Character-level language Model

: 문자열 시퀀스를 입력으로 받아서 다음 문자를 예측하는 모델.





하나의 출력값을 알기 위해서 처음부터 모든 Loss 값들을 알아야 함.

=> 계산량이 많아짐.

=> train 할 때 한 스텝을 일정 단위로 나눈 후, 서브시퀀스의 Loss들만 계산.

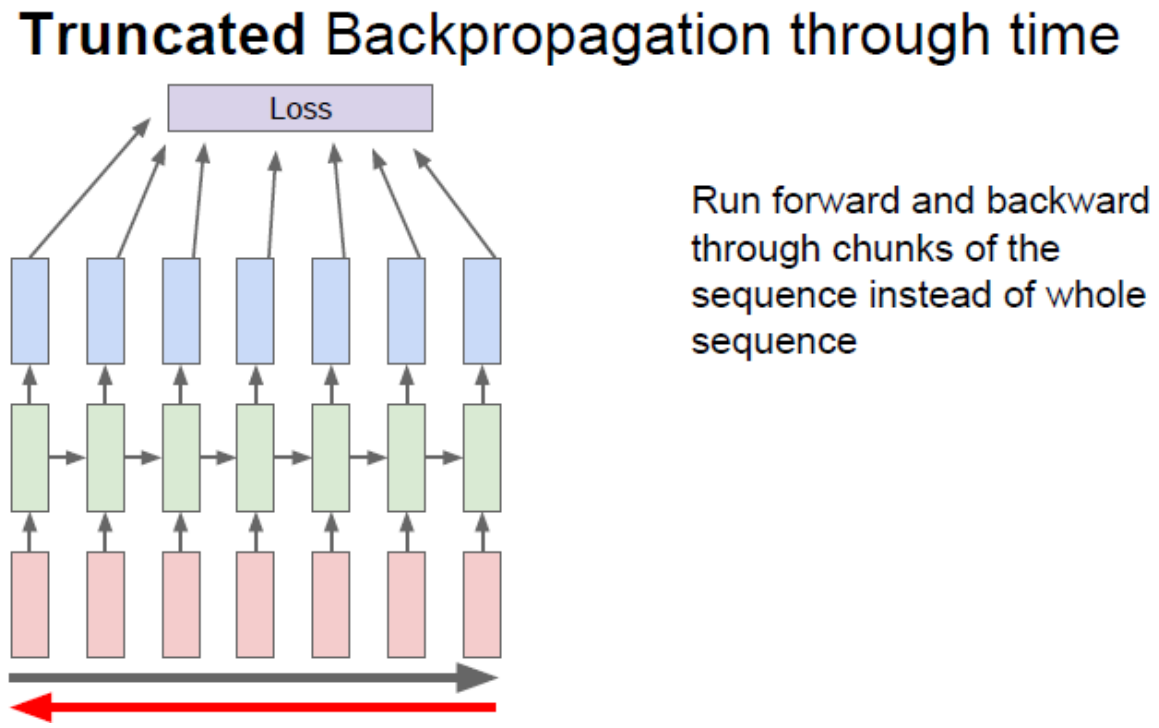
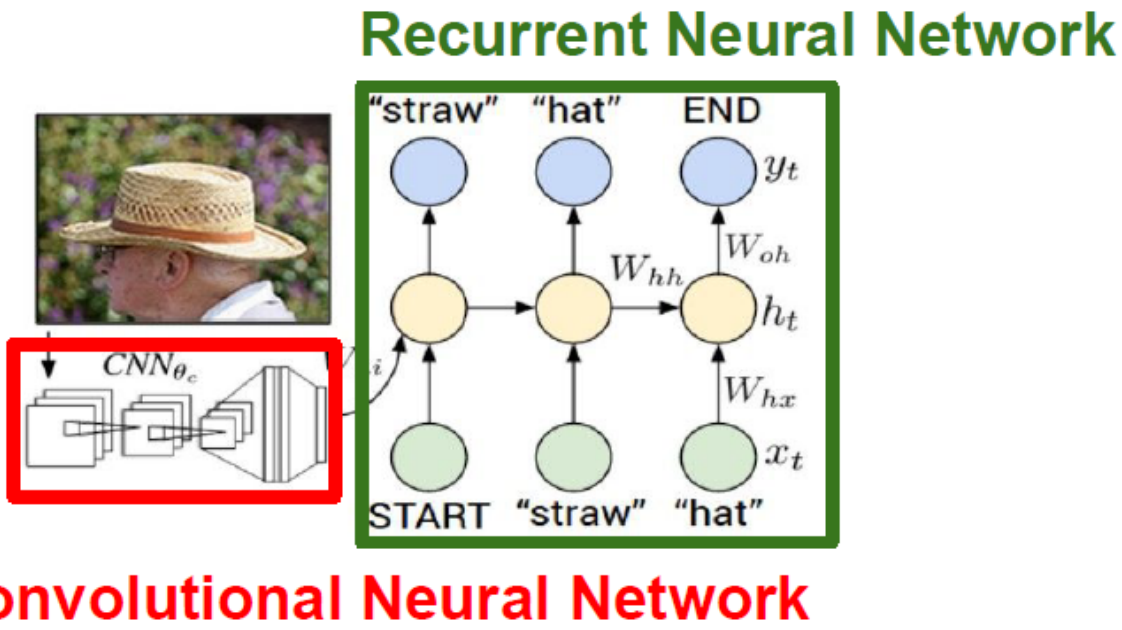
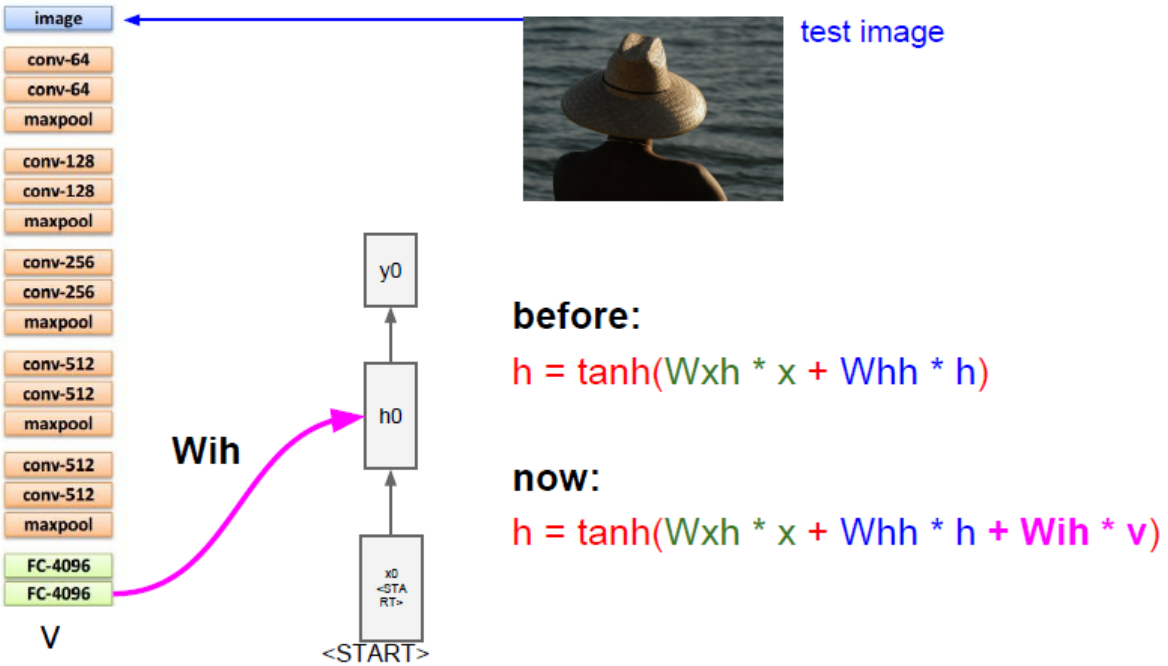


Image Captioning

: 하나의 사진을 보고 이 사진이 무엇인지 설명하는 문장을 만들어 내는 딥러닝 모델.

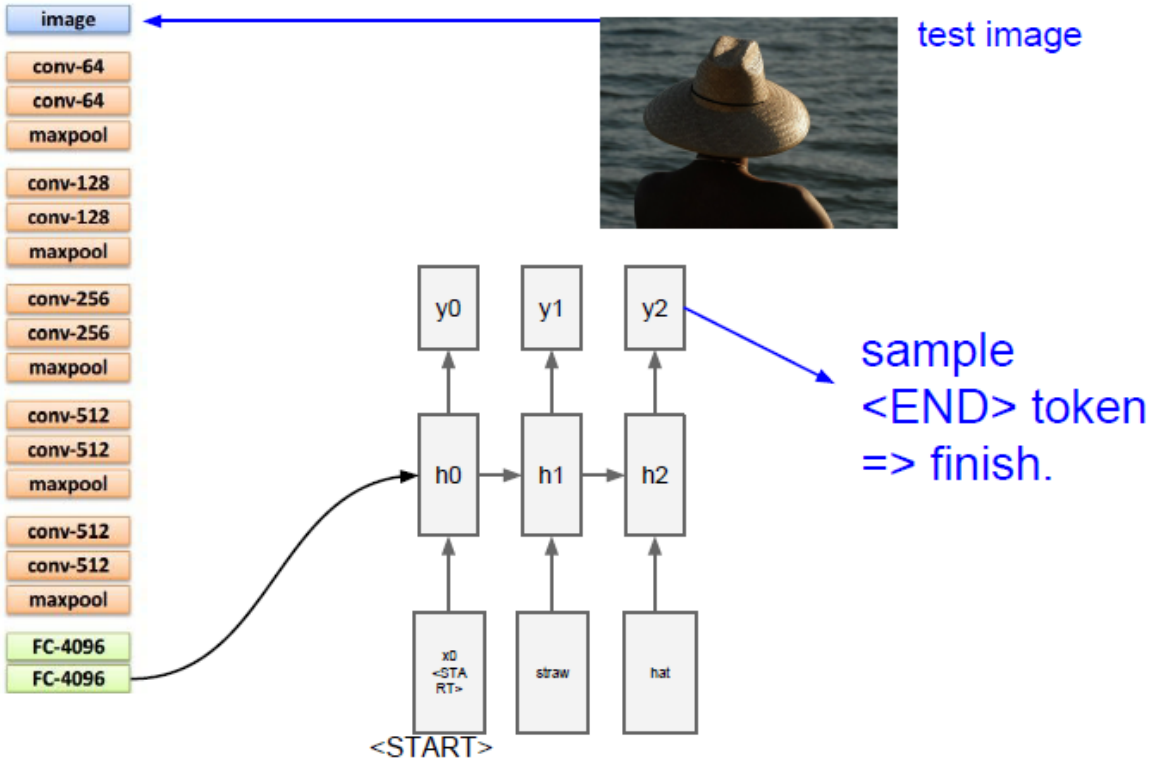


사진을 보고 CNN 구조를 통과한 결과가 RNN의 입력으로 들어가는 모습을 볼 수 있음.

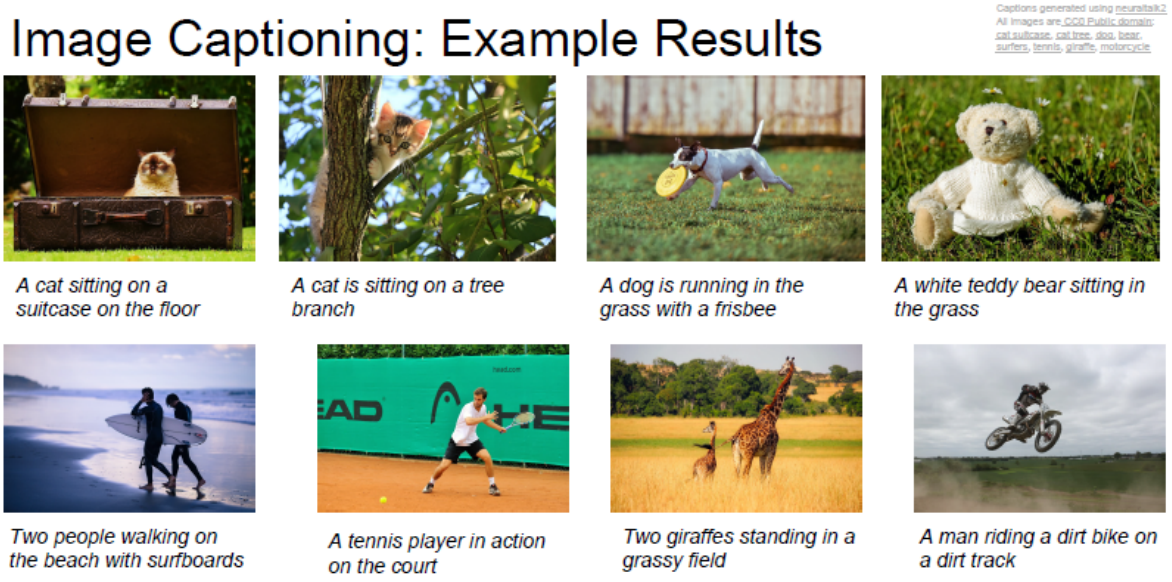


실제로 test Image가 들어가게 되면 Softmax를 통과하기 바로 전 Fully Connected Layer를 이용.

새로운 가중치 행렬을 추가하여 이미지 정보를 추가.



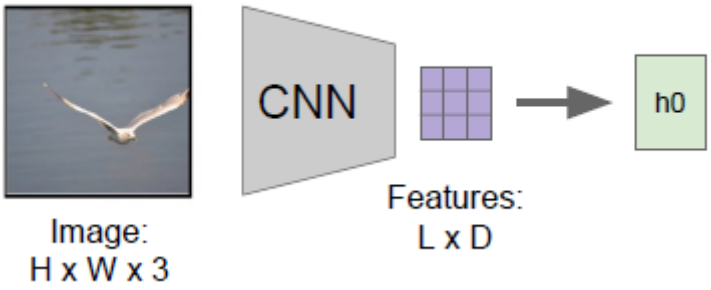
그리고 RNN 구조를 이용하여 문자열 결과를 출력.



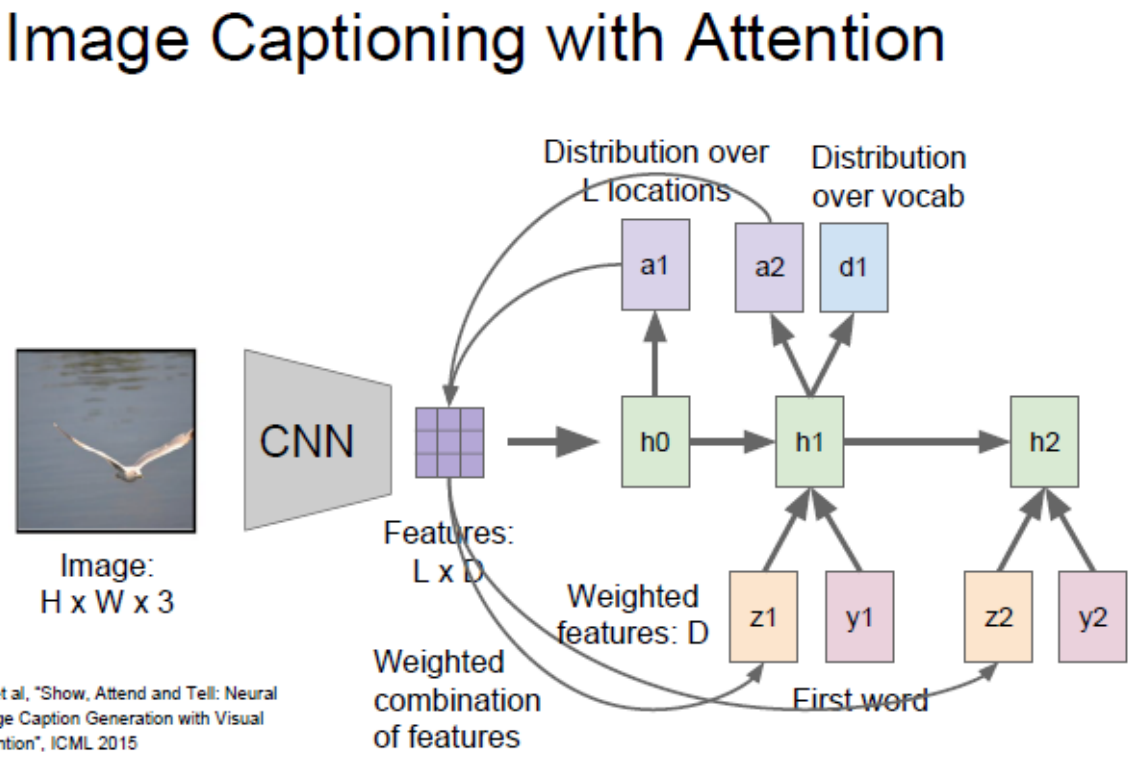
=> Supervised Learning.

Image Captioning with Attention

Input Image를 통해 공간 정보를 담고 있는 Grid of Vector 값을 얻어냄.



그 다음 Output을 살펴보면 총 2가지의 출력 존재.

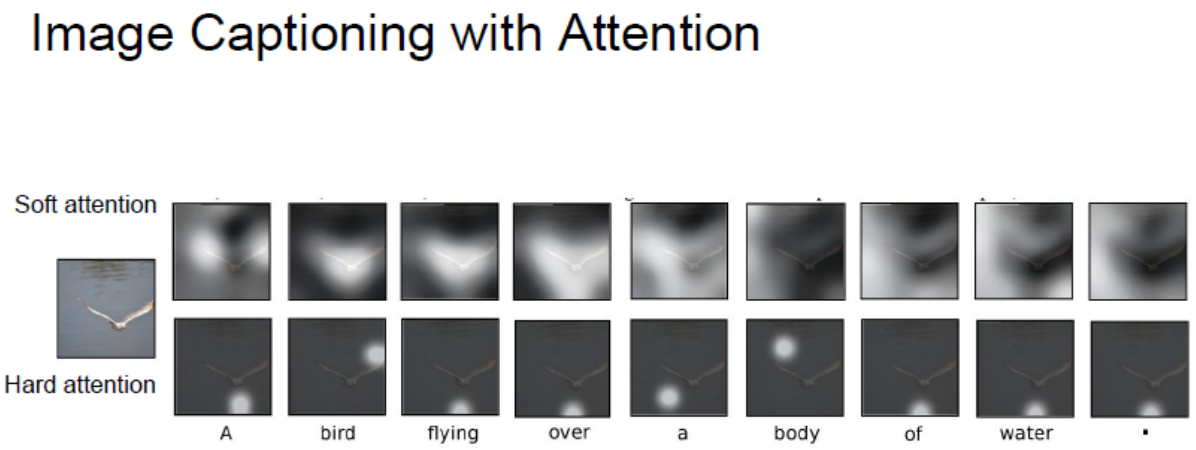


그림에서 a1, a2 등이 이미지의 분포를 나타낸 출력값, d1, d2 등이 단어의 분포를 나타낸 출력값.

$z=\sum_{i=1}^{L} p_i v_i$

a1의 결과와 Feature의 결과를 곱해서 z를 표현.

Image Captioning with Attention과정을 시각화하면



모델의 caption을 생성하기 위해 이미지의 attention을 이동하는 모습을 볼 수 있음.

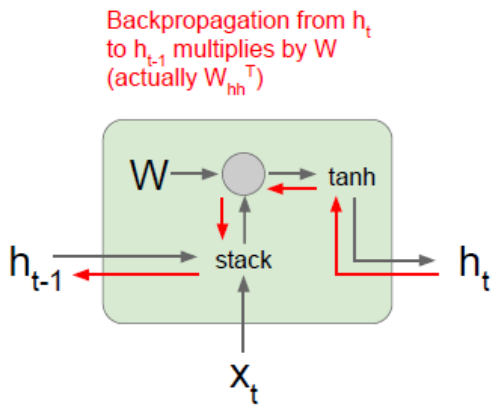
즉, 의미가 있다고 생각하는 부분에 스스로 attention을 집중하는 모습을 보인다.

VQA(Visual Question Answering)와 유사.

: 이미지와 질문을 동시에 던졌을 때, 답을 맞추는 과정.

모델이 깊어질 수록 다양한 문제들에 대해서 성능이 더 좋아지는 것을 확인할 수 있음.

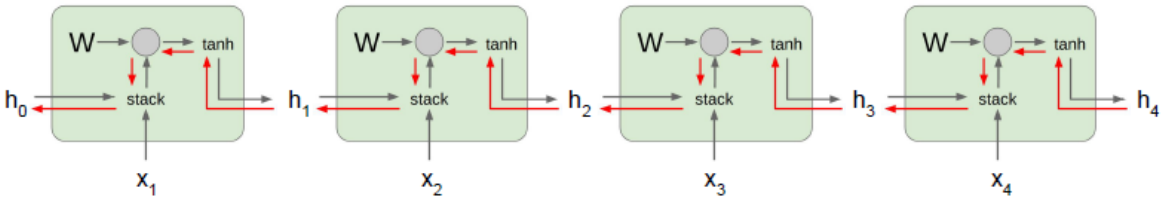
Train을 하기 위해서 Gradient값을 구해야 하는데, 먼저 Vanilla RNN Gradient Flow를 살펴보면



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

RNN 하나의 cell을 통과할 때 마다 가중치 행렬의 일부를 곱하게 되는 모습을 볼 수 있음.

하나의 cell이 아닌 전체적인 cell로 gradient를 구하는 모습을 살펴보면



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients**

Largest singular value < 1: **Vanishing gradients**

많은 W 행렬들이 개입하게 되어 계산량이 늘어나는 것을 확인할 수 있음.

문제점이 생기는데 Weight 행렬에서 특이값의 최대값에 따라 Gradient값이 폭발적으로 증가할 수도, 너무 작게 감소할 수도 있는 문제점이 발생.

이러한 문제점을 해결하기 위해 나온 새로운 RNN 구조 **=> LSTM**

LSTM

:Long Short Term Memory에 약자로 RNN의 새로운 구조.

Vanilla RNN

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

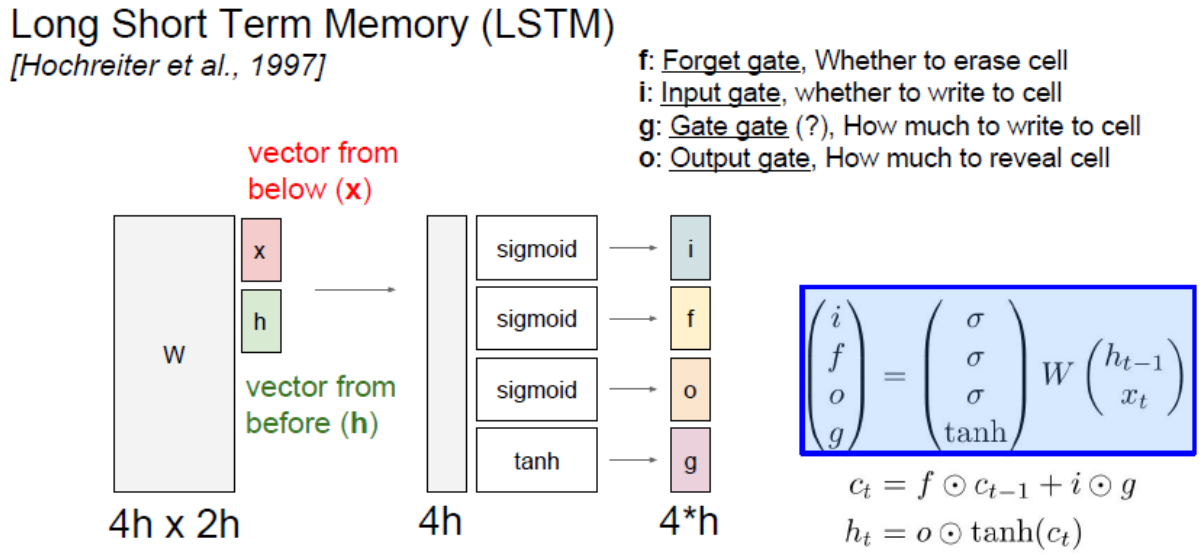
LSTM

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

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

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

기본적인 RNN 구조와 비교를 해보면, LSTM은 두개의 hidden state가 존재하고 ct라는 내부에만 존재하는 변수가 존재.



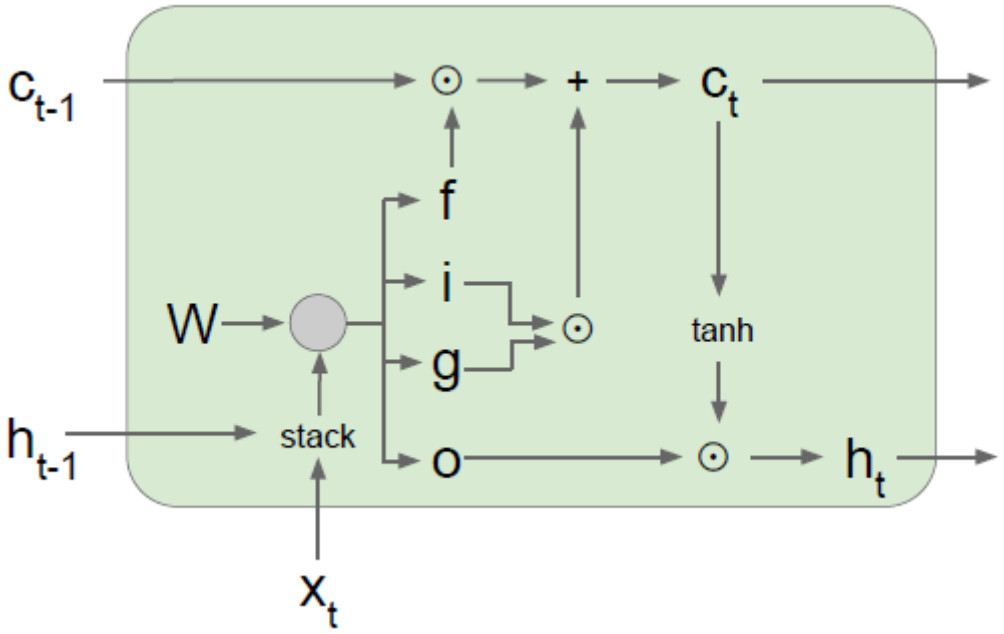
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

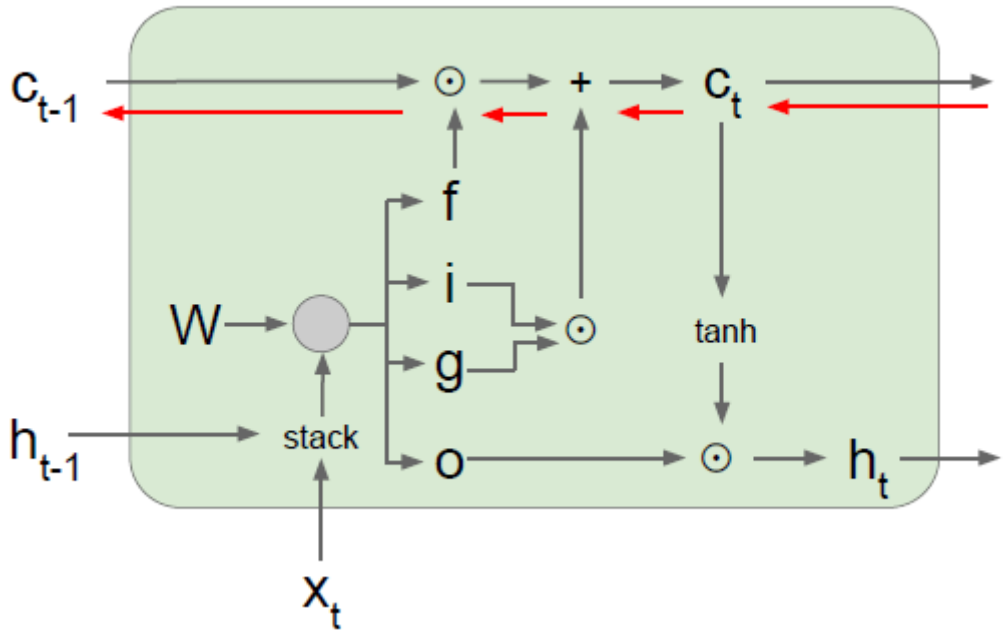
$$h_t = o \odot \tanh(c_t)$$

총 4가지의 gate

- *i - Input gate : cell*에 대한 입력 *(*xtxt에 대한 가중치*)*
- *f - Forget gate*
- *o - Output gate :*
- ct를 얼마나 밖으로 드러낼 것인지
- *g - Gate gate : Cell*을 얼마나 포함 시킬 것인지



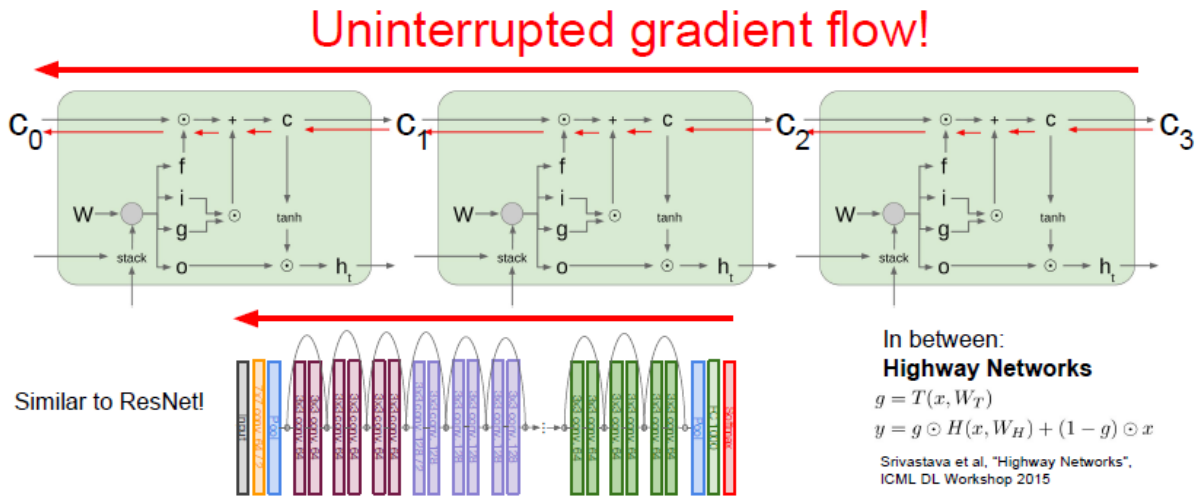
여기서 Gradient Flow 과정을 살펴보면



하나의 cell에서 Weight행렬을 거치지 않고도 cell 단위에서 gradient 계산을 할 수 있음.

여러 cell을 거치더라도 계산량이 크게 증가하지 않는다는 것.

즉, forget gate와 곱해지는 연산이 행렬 단위의 곱셈 연산이 아니라 element-wise (원소별 연산)이기 때문에 계산량이 크게 증가하지 않음.



In between:
Highway Networks

$g = T(x, W_T)$

$y = g \odot H(x, W_H) + (1 - g) \odot x$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015