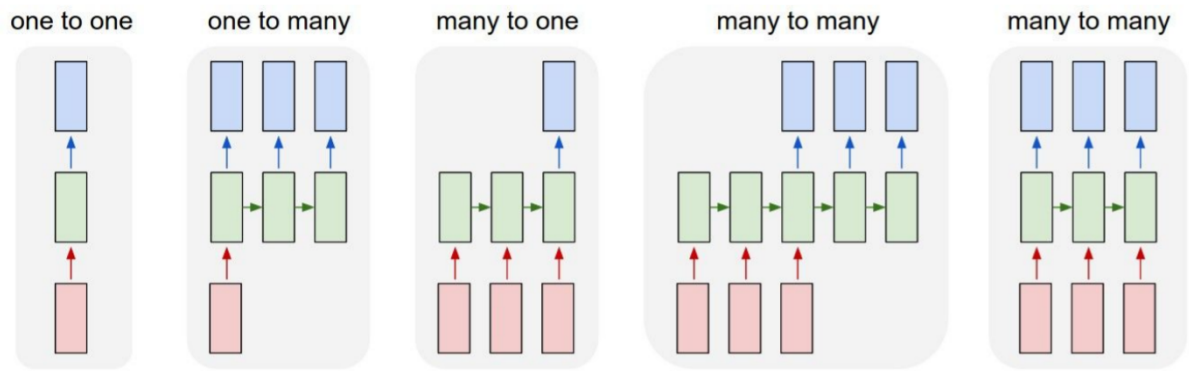
Lec 10. Recurrent Neural Networks

Recurrent Neural Network

: sequence model(input과 output을 sequence 단위로 처리하는 model)

- one to one: 가장 기본적인 형태. 하나의 input과 하나의 output
- one to many: 하나의 input에 대해 여러 output
  - image captioning: 하나의 사진에 대해 여러 단어 출력
- many to one: 여려 input에 대해 하나의 output
  - 감정 구별
- many to many: 여러 input에 대해 하나의 output
  - 비디오에 대해 프레임 단위로 classification, 기계 번역



RNN 기본 구조, 식

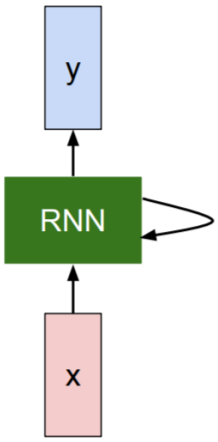
Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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

new state / old state / input vector at some time step

some function with parameters W



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

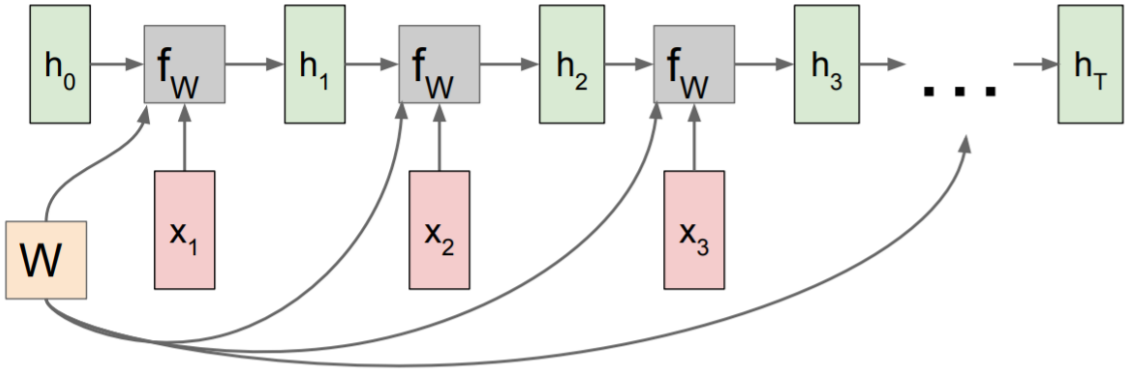
$$\downarrow$$

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

$$y_t = W_{hy}h_t$$

- 모든 함수와 parameter 값을 모든 시간에서 동일하게 이용

RNN Computational Graph



- RNN 기본 구조를 쌓아올려 여러 input 받음
- 각 step에서는 같은 weight 사용

- many to many
  - gradient: 각 step에서 weight에 대한 gradient의 총합
  - loss: 각 step에서 구한 y_t에 대한 loss를 모두 더한 값
- many to one: 최종 hidden state에서만 결과값이 나옴
- one to many: 하나에 대해 여러 output이 나옴
- sequence to sequence: many to one + one to many
  - 기계 번역
  - encoder(many to one) + decoder(one to many)

ex) Character-level Language Model

- 문자열을 예측하는 언어 모델
- One-hot encoding
  - 하나의 값만 선택하여 사용하는 방법
  - h = [1, 0, 0, 0], e = [0, 1, 0, 0], l = [0, 0, 1, 0], o = [0, 0, 0, 1]과 같이 4개의 값 중에 하나만 꼽아서 사용하는 것
- Sampling
  - softmax를 추가해 확률적으로 나올만한 것을 꼽는 것

RNN Truncated Backpropagation

- RNN은 앞선 hidden state를 이용하기에 compute time과 training time이 오래 걸림
- 많은 memory 사용
- 따라서 batch 별로 나누어 loss 계산하고 train하는 방식 사용 → RNN truncated backprop

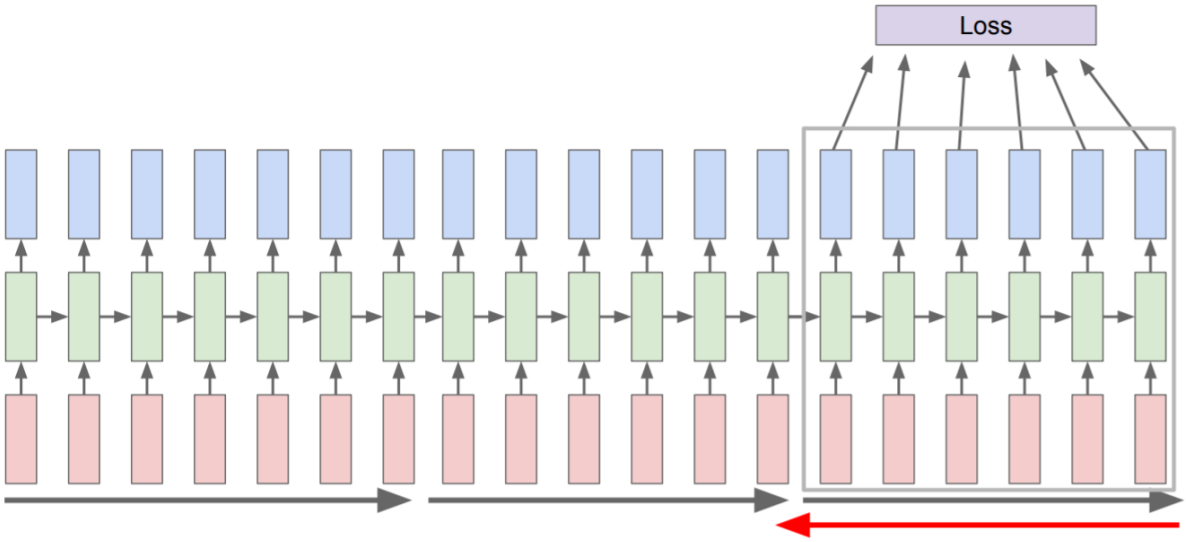
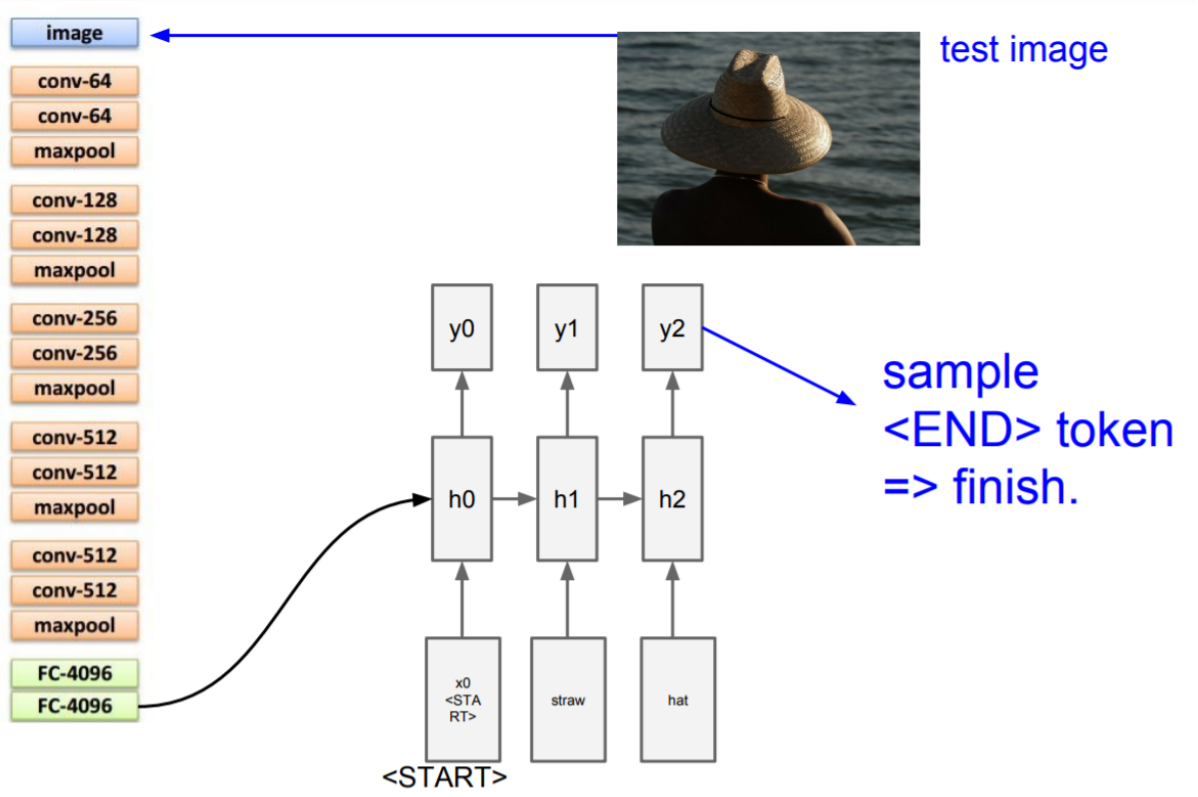
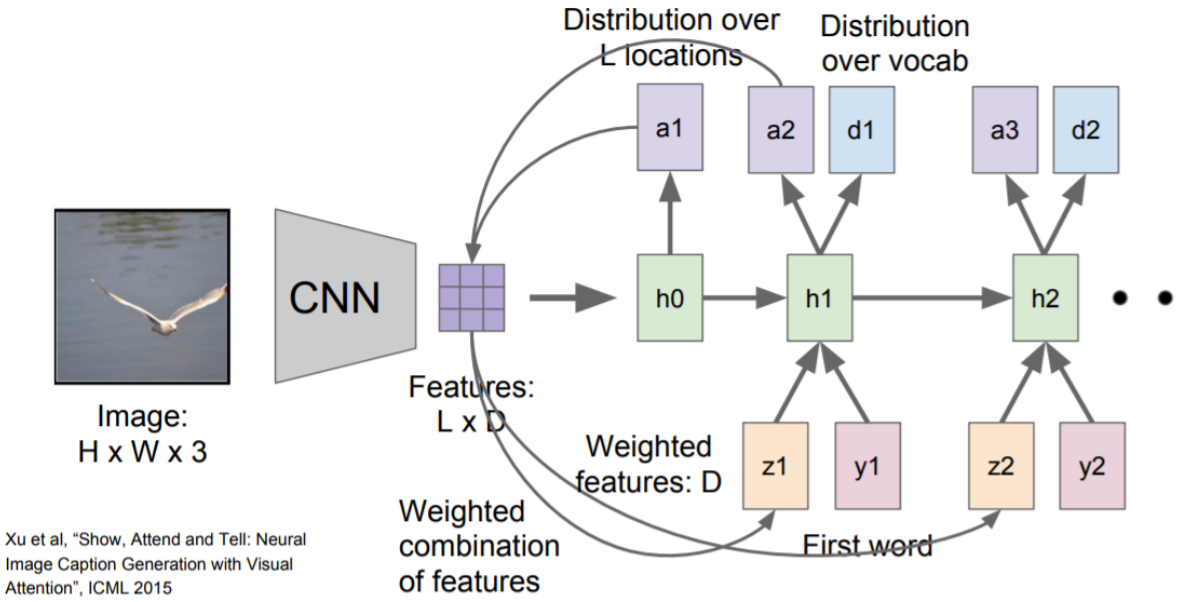


Image Captioning

- CNN의 output을 RNN의 input으로 사용하고 image에 대한 caption을 만들어내는 것



- 해당 방식은 input image를 모두 사용하여 전체적으로 captioning을 하는 방식으로, Top-Down Approach라 함
- 이미지의 다양한 부분을 폭넓게 볼 수 있음
- but, 이미지의 디테일한 부분에는 취약함
- attention 기법은 이미지의 각 부분에서 특징을 받아와 caption을 넣어 좀 더 자세함 → Bottom-Up Approach



- 각 벡터가 공간정보를 가지고 있는 grid of vector(LxD)를 만듦
- soft attention: 0~1 범위값 사용
- hard attention: 0 or 1의 값 사용

Multilayer RNN: 여러개의 hidden layer를 사용하는 것

Multilayer RNNs

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $\quad W^l \ [n \times 2n]$

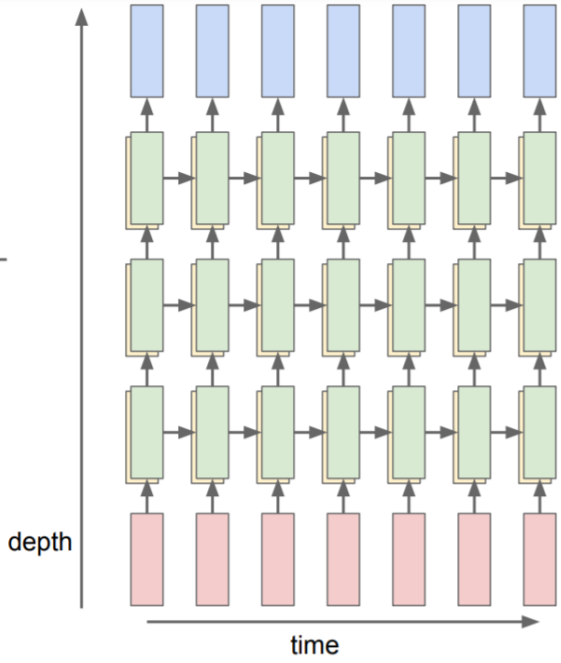
LSTM:

$W^l \ [4n \times 2n]$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

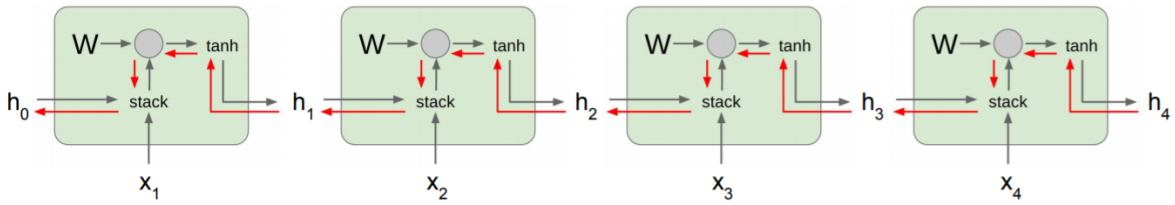
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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



Vanilla RNN Backprop

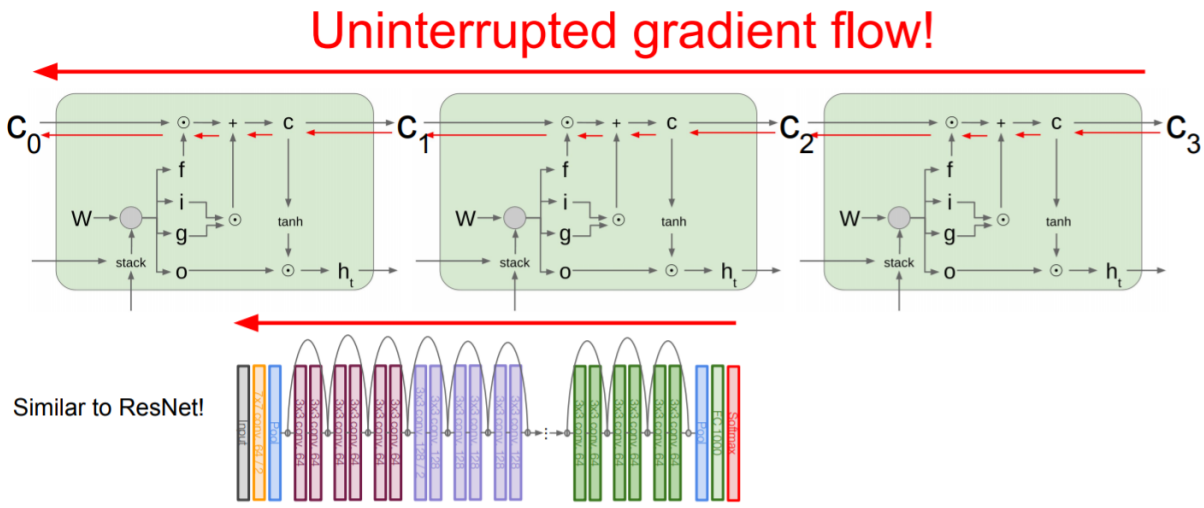
- exploding gradients 문제: gradient가 너무 커지는 것 → gradient clipping으로 해결
- vanishing gradients 문제: gradient가 너무 작아지는 것 → RNN 구조 자체를 바꿔야 함



LSTM: RNN의 vanishing gradient 문제의 solution

- 4개의 gate 이용
  - i: input gate → sigmoid
    - 현재 input을 얼마나 반영할 것인가
  - f: forget gate → sigmoid
    - 이전 input을 얼마나 기억할 것인가
  - o: output gate → sigmoid
    - 현재 cell 안에서의 값을 얼마나 보여줄 것인가
  - g: gate gate → tanh
    - input cell을 얼마나 포함시킬 것인가, 얼마나 학습시킬 것인가(weight 관련)
- ifo는 잊기 위해서는 0에 가까운 값, 기억하기 위해서는 1에 가까운 값을 나타내기 위해 sigmoid 이용
- g는 강도와 방향을 나타냄
- 🙂
  - forget gate의 elementwise multiplication이 matrix multiplication보다 계산적으로 효율적
  - forget gate값을 곱하여 사용하므로 같은 weight를 곱해주던 위 형태와 다르게 input에 따라 다른 값을 곱하기에 vanishing/exploding 문제를 피할 수 있음

- ResNet과 비슷한 backprop 형태



GRU: LSTM의 변형

- LSTM과 비슷한 성능을 가짐
- GRU가 LSTM보다 단순한 구조 → 빠름 but, 이를 비교하는 것보다는 실험적으로 더 빠른 모델을 이용하는 것이 더 좋음
- GRU, LSTM 이외의 다른 방식을 찾아보는 노력이 있었으나 이들보다 확연히 뛰어난 모델이 나오지 않았음