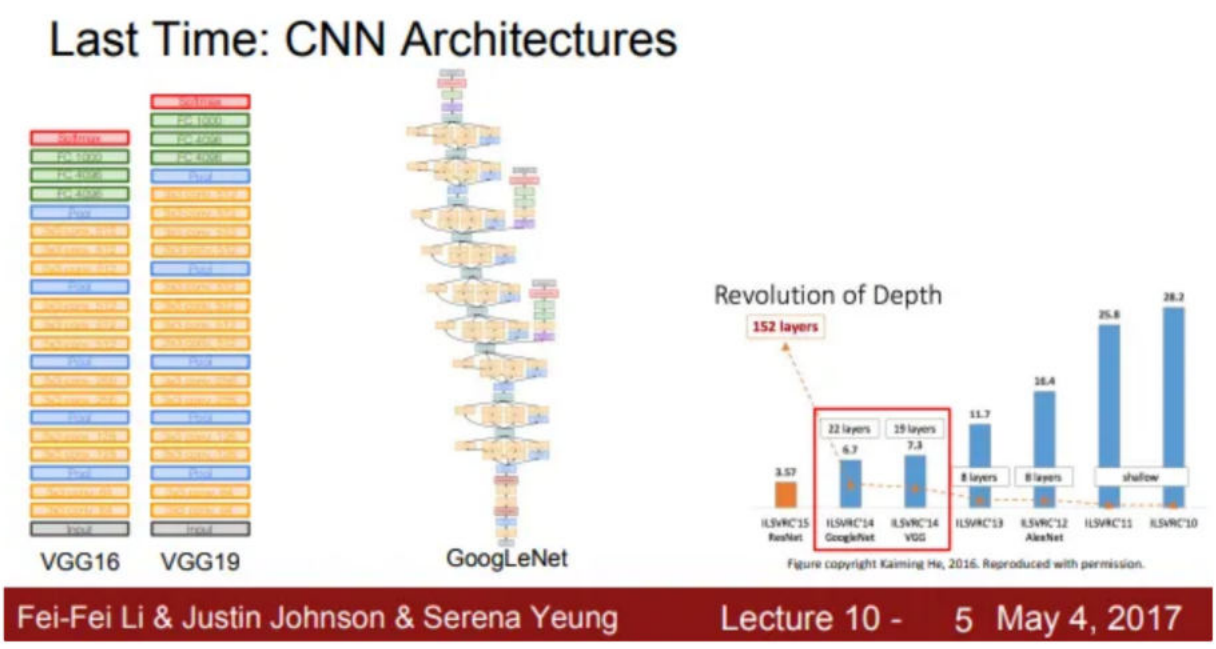
exam 개인학습

Lec 9

AI summary Try on this page

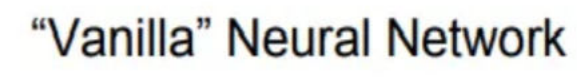
Type something...

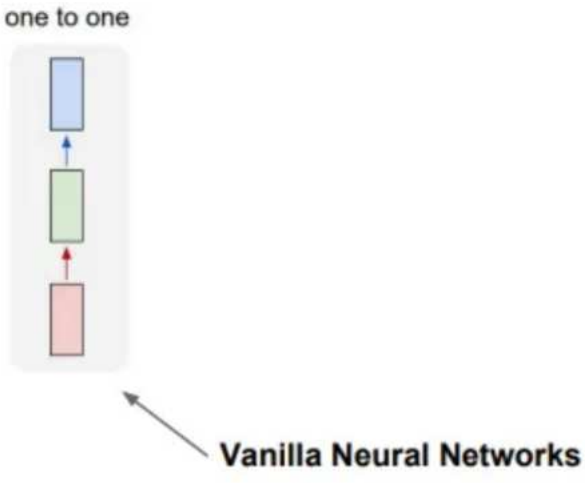
01) Recurrent Neural Networks(RNN)



- batch norm의 등장 전까지는 22-layer도 매우 도전적인 과제였다.

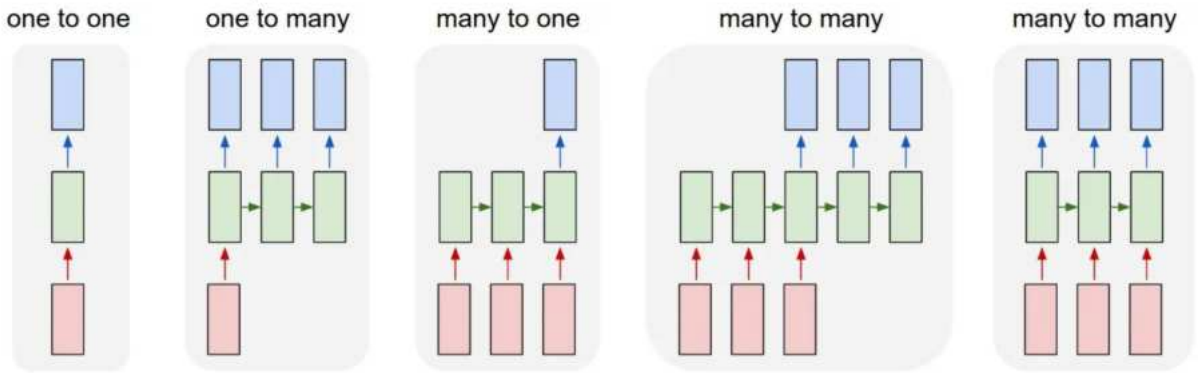
01_Vanilla Neural Network





- `Vanilla Neural Network` 은 고정된 크기의 vector를 입력으로 받아 고정된 크기의 vector를 출력한다.

02_Recurrent Neural Network example



1. **one to one**

   RNN이 없는 Vanilla mode이며, fixed-sized input을 fixed-sized output을 처리합니다.

2. **one to many**

   Sequence output (e.g. Image Captioning, image -> sequence of words)

3. **many to one**

   Sequence Unit (e.g. Sentiment(감정) Classification, sequence of words -> sentiment)

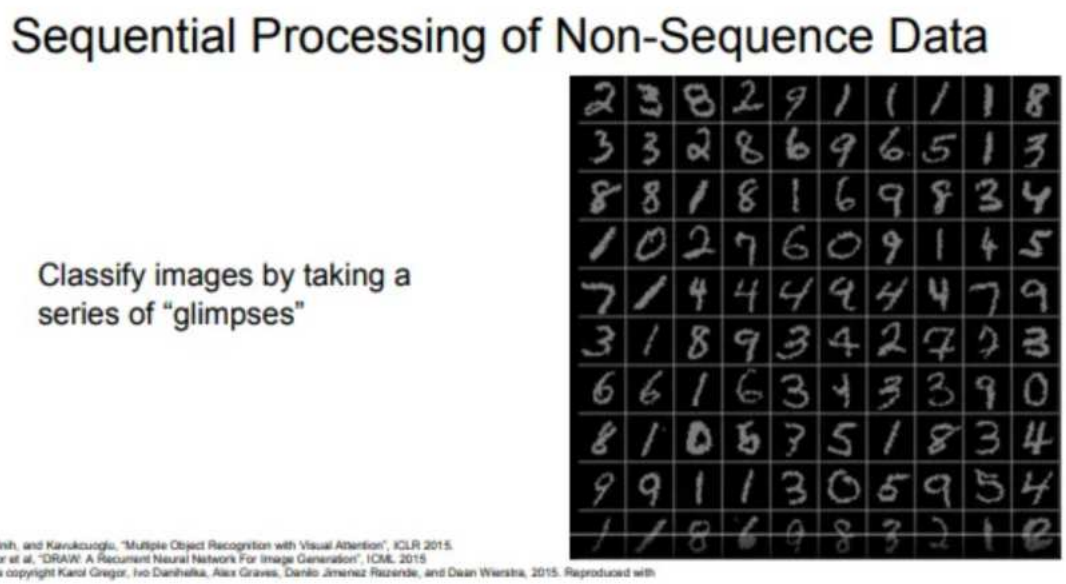
4. **many to many**

   Machine translation (e.g. Machine Translation, seq of words in English -> seq of words in French)

5. **many to many**

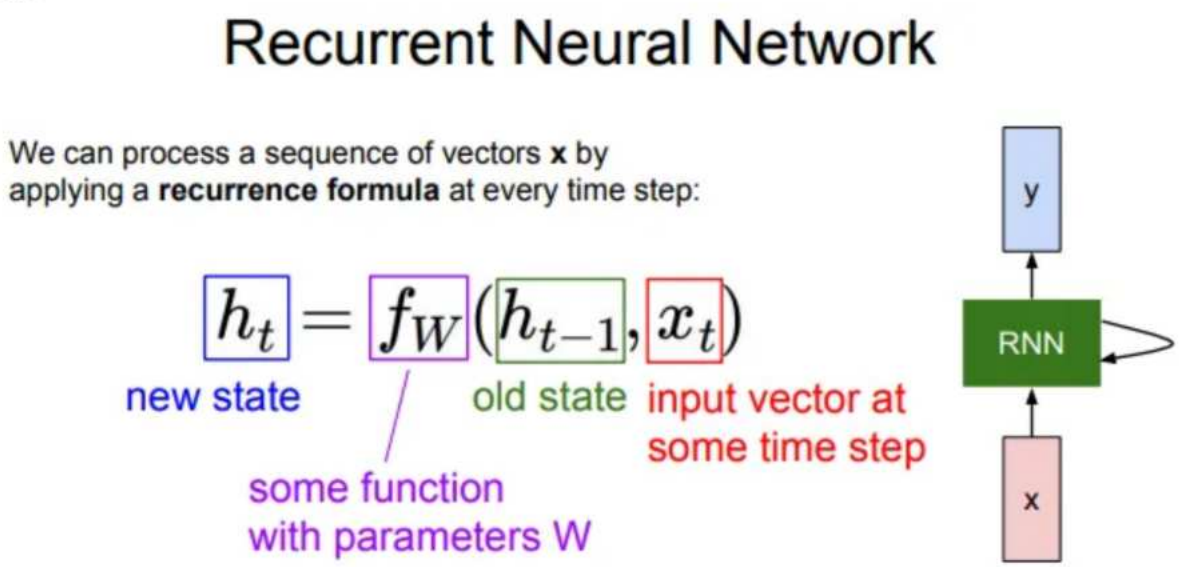
   Synced sequence input and output (e.g. Video Classification on frame level)

03_Sequential Processing of Non-Sequence Data



- fixed-size의 input일 때도 RNN을 사용할 수 있다.
- 입력 이미지의 label을 feed-forward pass 한 번으로 결정하는 것이 아니라 RNN network를 통해서 **이미지의 여러 부분을 조금씩 살펴본 후 최종적으로 판단한다.**

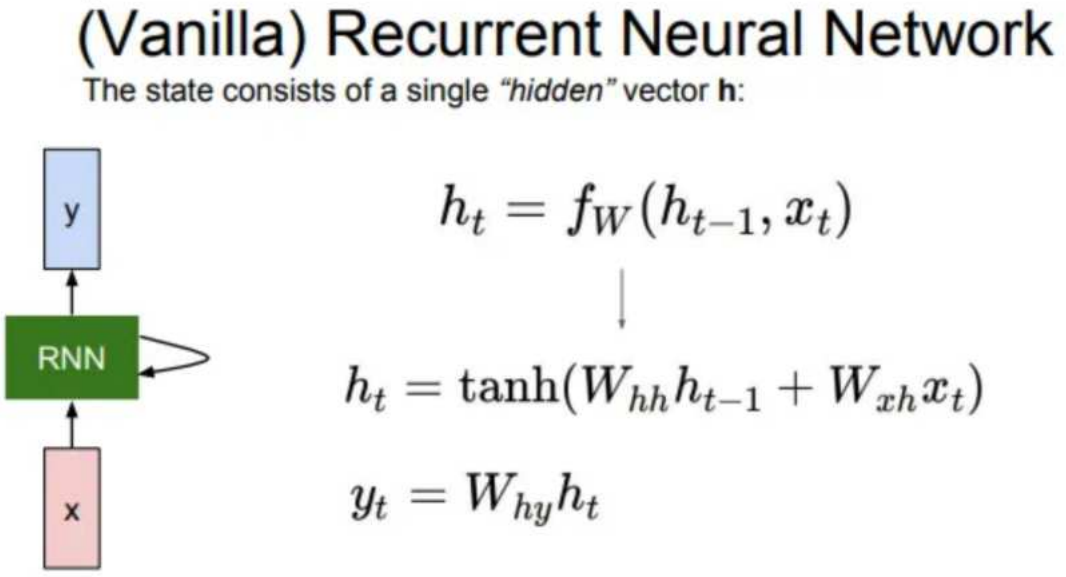
04_(Vanilla) RNN



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

1. Sequence vector x를 input으로 받는다.
2. hidden state를 update 한다.
3. 출력 값을 내보낸다.

- 출력 값을 가지려면 FC-layer를 추가해야 한다.
- **hidden_state를 기반으로 출력 값을 정한다.**
- **함수 f와 parameter W는 매 step에서 동일하다.**

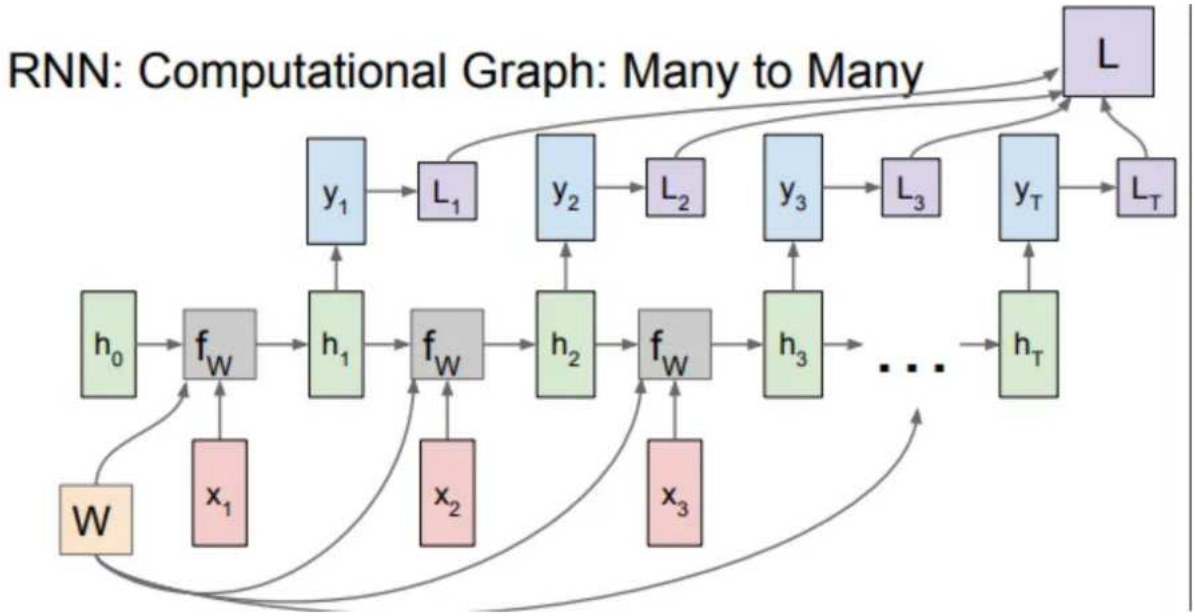


$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

- non-linearity를 위해 tanh를 사용한다.

1. Computational Graph(Many to Many)

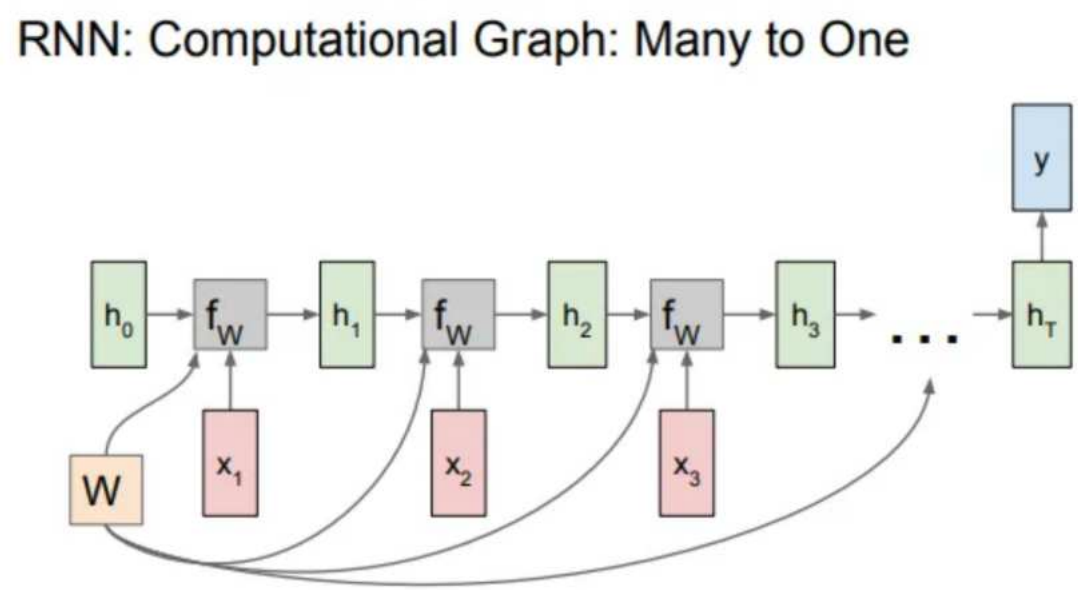


$$h_1 = f_w(h_0, x_1)$$

$$h_2 = f_w(h_1, x_2)$$

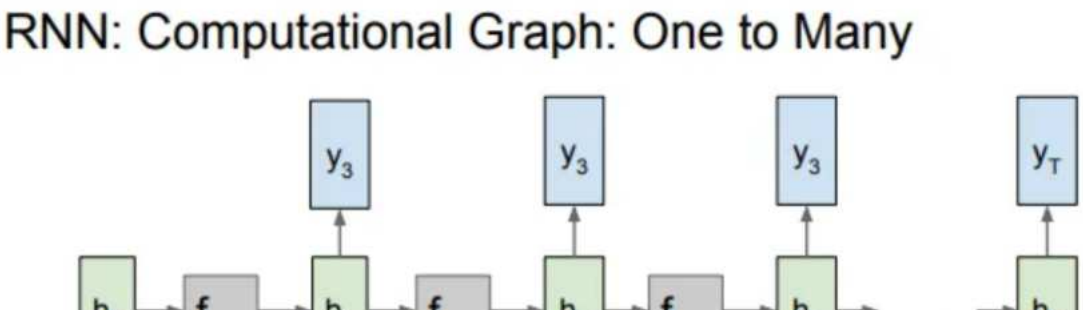
- 이 과정을 반복하여 hidden_state들 구해준다.
- 중요한 점은 동일한 가중치 행렬 W가 사용된다는 점이다.
- `dLoss/dW` 를 계산하려면 각 step에서의 W에 대한 gradient를 전부 계산한 뒤에 이 값들을 모두 더해준다.
- Loss는 softmax loss를 사용하였다. 최종 Loss는 각 loss들의 합이다.

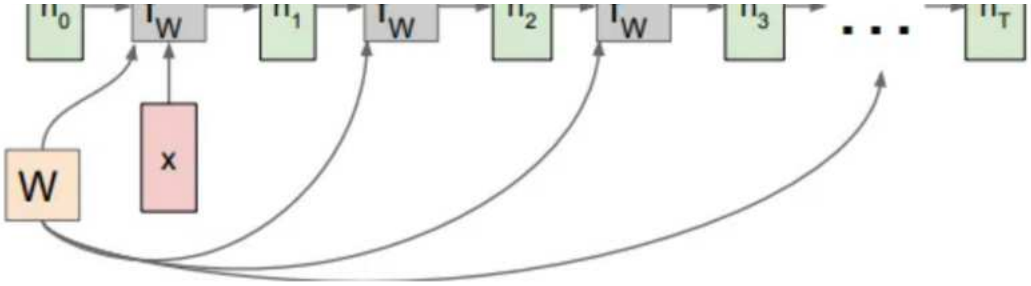
2. Computational Graph(Many to One)



- 최종 hidden_state에서만 output이 나올 것이다.
- 최종 hidden_state가 전체 시퀀스 내용에 대한 요약이기 때문이다.

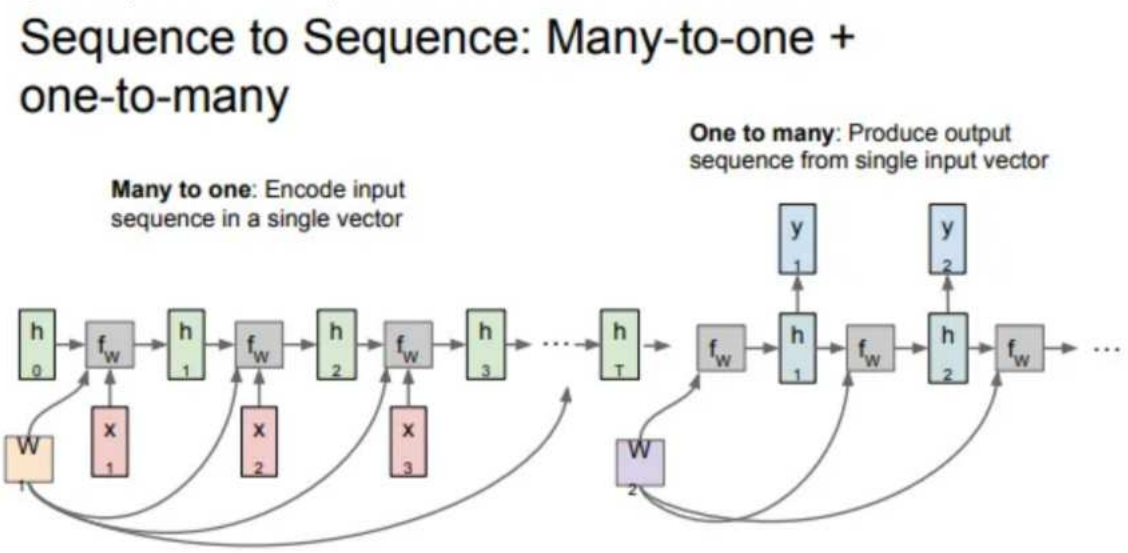
3. Computational Graph(One to Many)





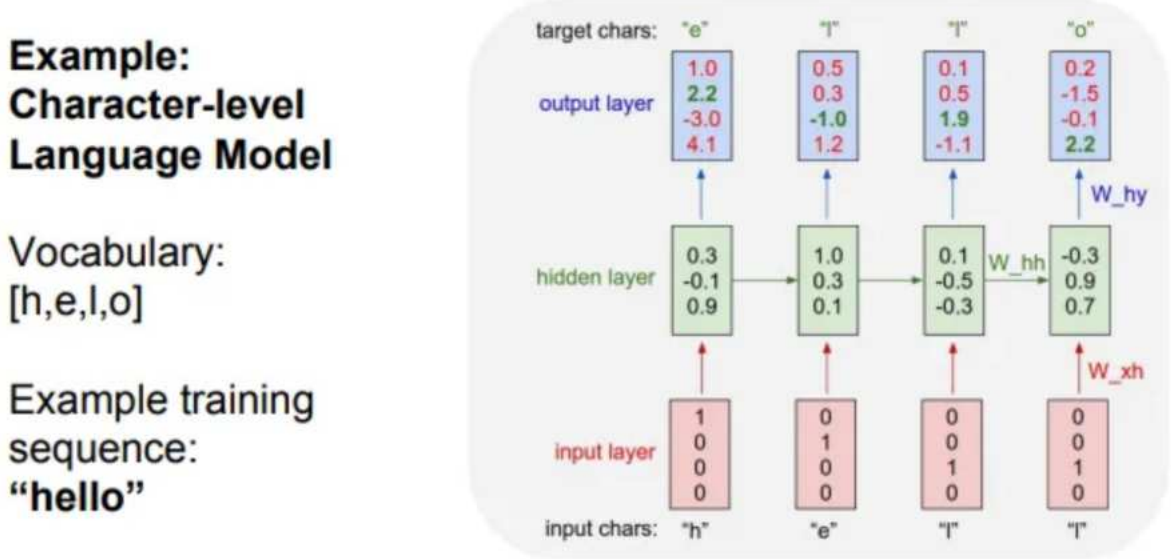
- 고정 입력의 모델은 initial hidden state를 초기화 시키는 용도로 사용된다.

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



- varied input을 받아서 varied output을 만드는 Seq2Seq모델을 한번 살펴보자.
- Encoder와 Decoder 두 부분으로 나누어져 있다.
- Encoder는 Many-to-one 구조로 예를 들어 영어 문장을 입력 받으면 전체 문장의 요약을 final hidden state에 저장한다.
- Decoder는 One-to-Mnay 구조로 Encdoer에서 요약된 final hidden state를 입력으로 하여 varied output을 만들어낸다.

05_Natural Language Model

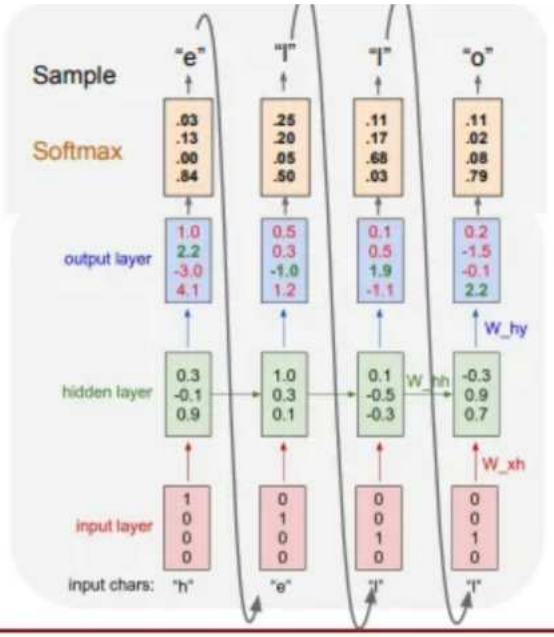


- 문자열을 읽어온 후 현재 문맥에서 다음 문자열을 예측해야 한다.
- 이번 예시의 입력은 'hello'이다.
- 우리가 예측하고 싶은 것은 ''ello' , 마지막이 'o'로 끝나는 것이다.
- 현재 문자가 4개 있기 때문에 `one-hot encoding` 을 통해서 각각의 문자를 벡터로 만들어 준다.
- 그 후 RNN forward pass를 통과하여 hidden, output을 만들어낸다.

Example: Character-level Language Model Sampling

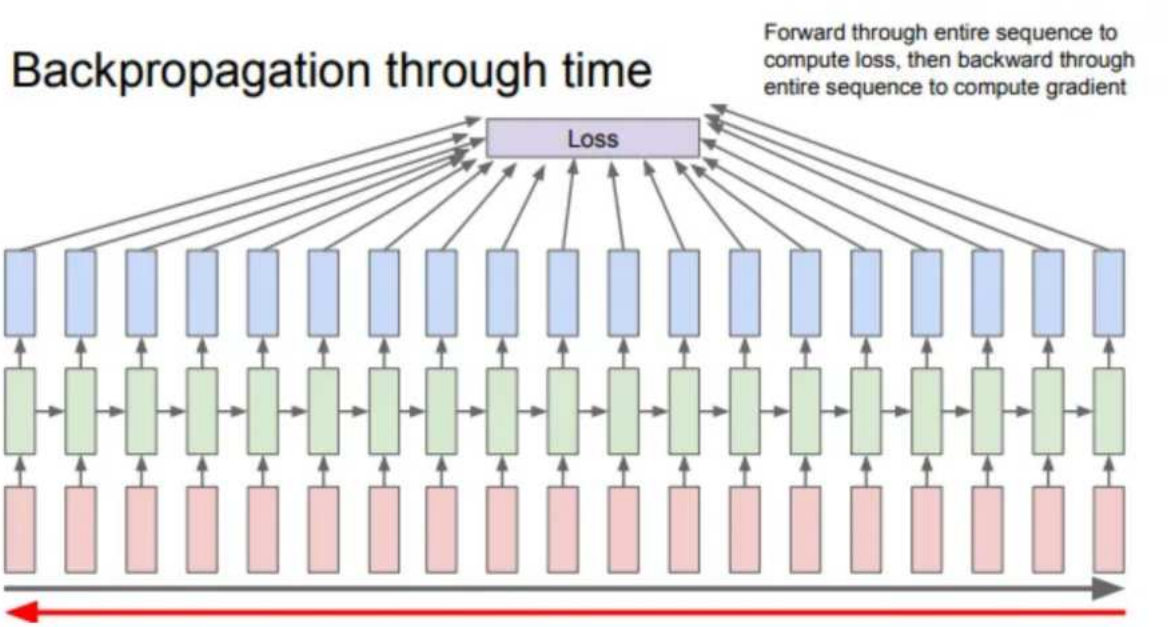
Vocabulary:
[h,e,l,o]

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

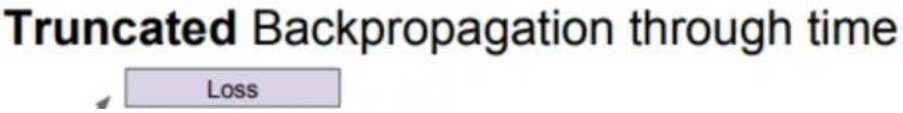


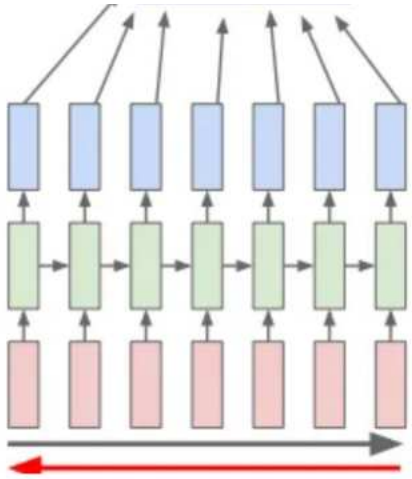
- test time 때는 모델이 스스로 무장을 생성한다.
- **첫 번째 output을 다음 스텝의 네트워크 입력으로 넣어준다.**
- time-step마다 확률 분포에서 문자를 하나씩 뽑아낸다.

06_Truncated backpropagation through time



- 자연어 처리 문제는 문장이 매우 길다면 gradient를 계산하는데 매우 느릴 것이고, 메모리도 부족할 것이다.

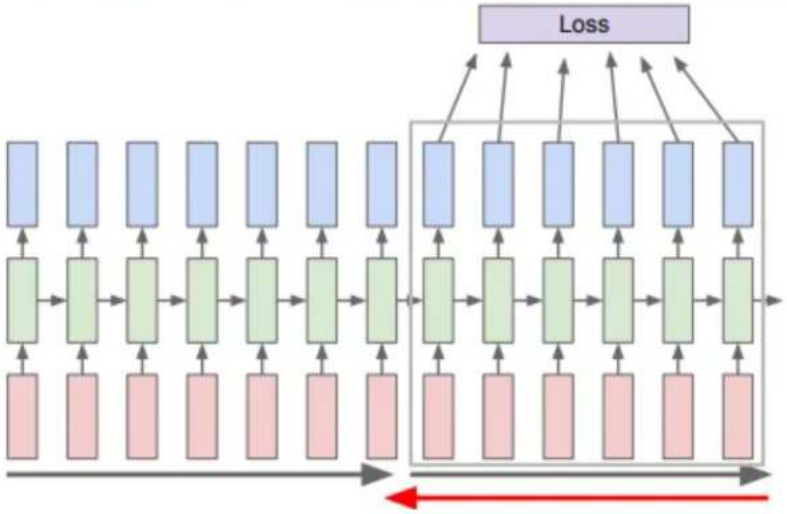




Run forward and backward through chunks of the sequence instead of whole sequence

- 100 step 정도로 잘라서 loss를 계산한 후 gradient step를 진행한다.

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

- 이전 batch에서 계산한 hidden state는 유지한 상태로 forward pass를 진행한다.
- 그리고 gradient는 현재 batch에서만 진행한다.
- **이전 batch에서 hidden states를 가져와 forward pass를 수행하고 backpropb은 현재 batch만큼만 진행한다.**
- stocastic gradient descent의 sequence data 버전으로 볼 수 있다.

07_RNN Language model - latent structure(숨겨진 구조)

- 셰익스피어의 소설을 한번 학습 input으로 사용해보자.

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

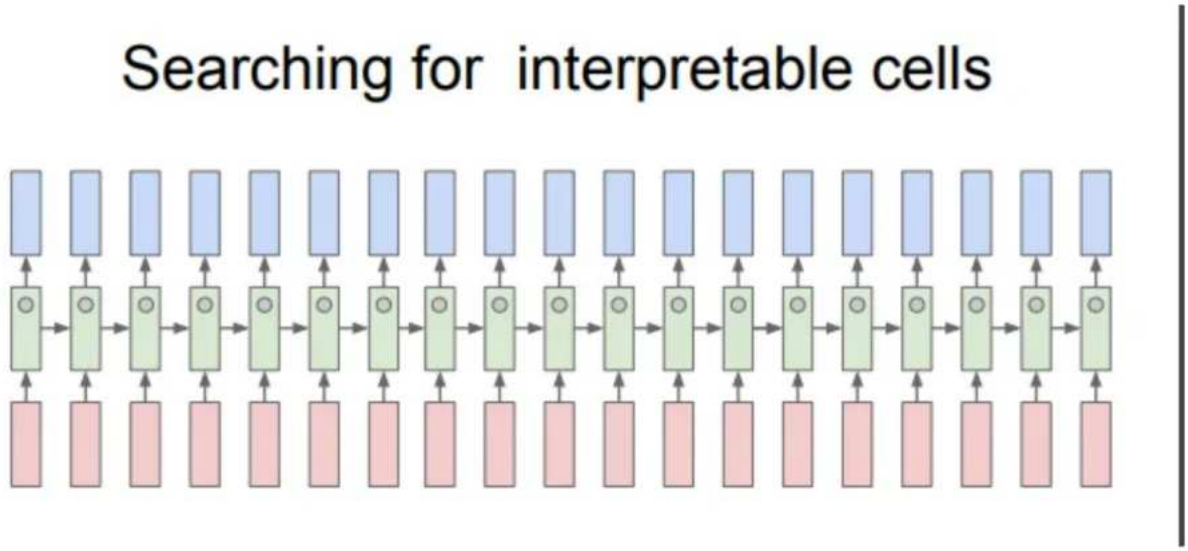
```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

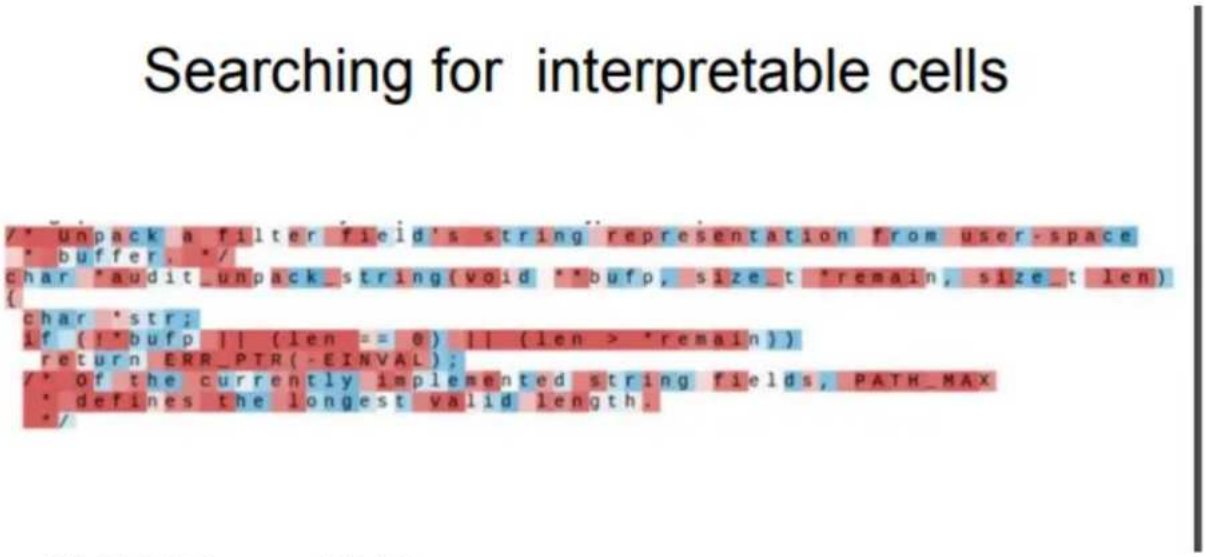
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

- 처음에는 정말 형편 없지만, 학습을 계속하다 보면 점점 소설과 비슷해지는 것을 확인할 수 있다.
- 모델은 학습 과정 속에서 sequence 데이터의 숨겨진 구조를 알아서 학습한다.
- 그렇다면 모델이 어떤 방식으로 학습을 하는 것일까?

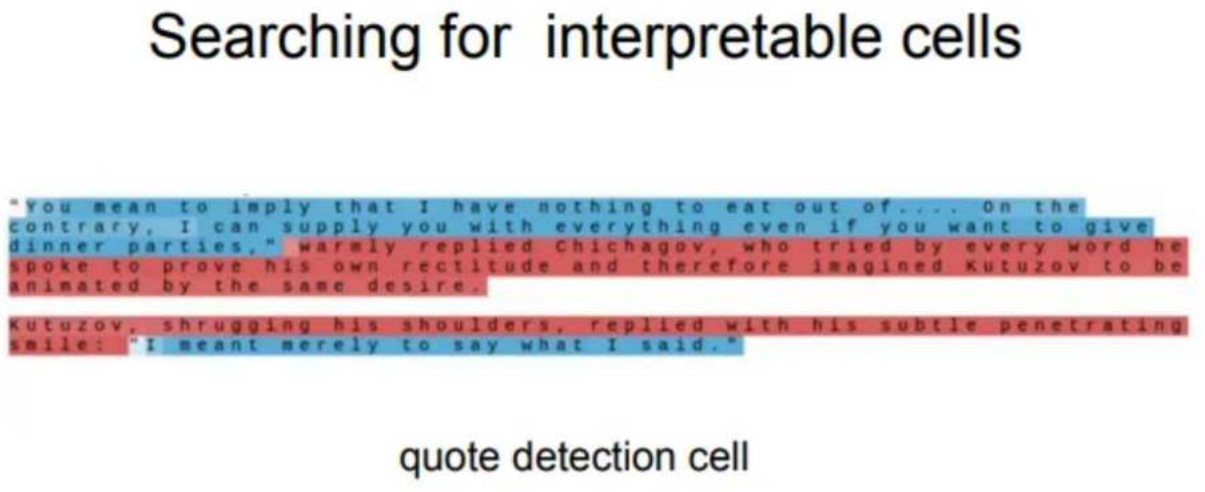
08_Searching for interpretable cells



- 모델은 hidden vetocr를 계속 업데이트를 한다.
- 모델을 이해하기 위해서 hidden vecotr의 값을 뽑아서 한번 관찰해보기로 하였다.



- 각 색깔은 sequence를 읽는 동안에 앞서 뽑은 hidden vector의 값을 의미한다.



- "" 이 각각 다른 색깔의 블록으로 표시되는 것을 알 수 있다.

Searching for interpretable cells

Cell sensitive to position in line:

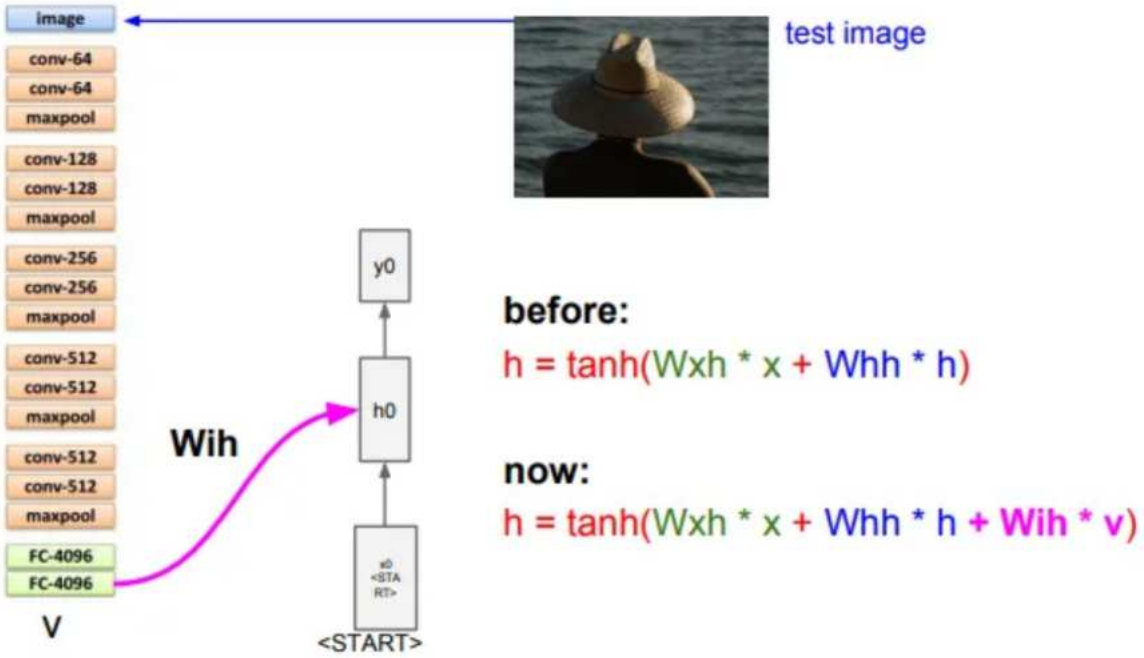
```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

line length tracking cell

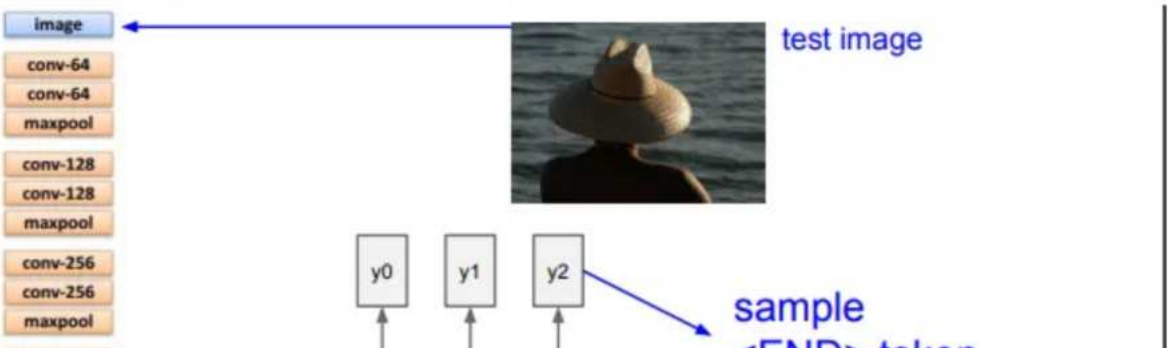
Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

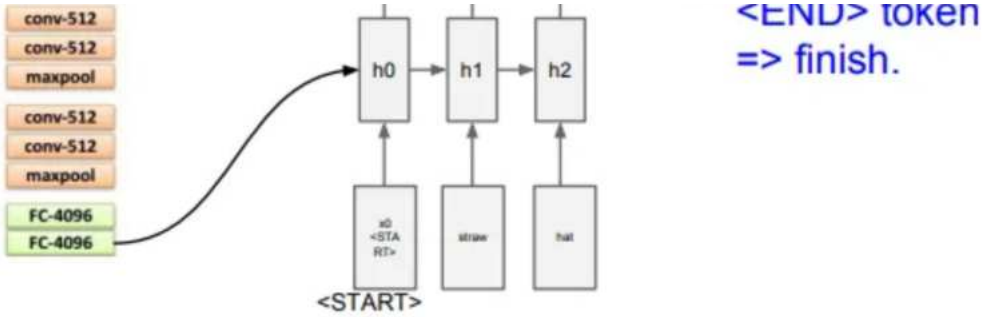
- 줄바꿈을 위해 현재 줄의 단어 갯수를 세는 듯해 보이는 cell도 발견하였다.
- 처음에는 0으로 시작하였다가 줄이 길어지면 점점 빨간색으로 변한다.
- 줄바꿈이 되면 다시 0으로 리셋된다.
- **결국 모델은 입력 데이터의 구조를 학습하게 되었다!**

09_RNN example of Image captioning



- image를 CNN에 넣어서 사진 데이터를 요약시킨 4,096 dim vector를 만든다.
- hidden state로 4,096 dim vector를 사용한다.

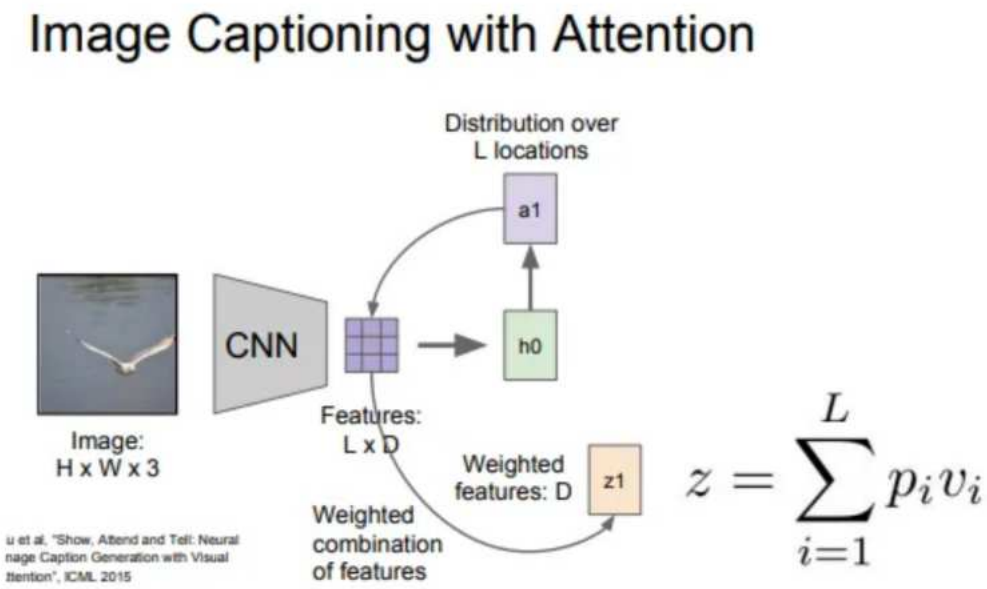




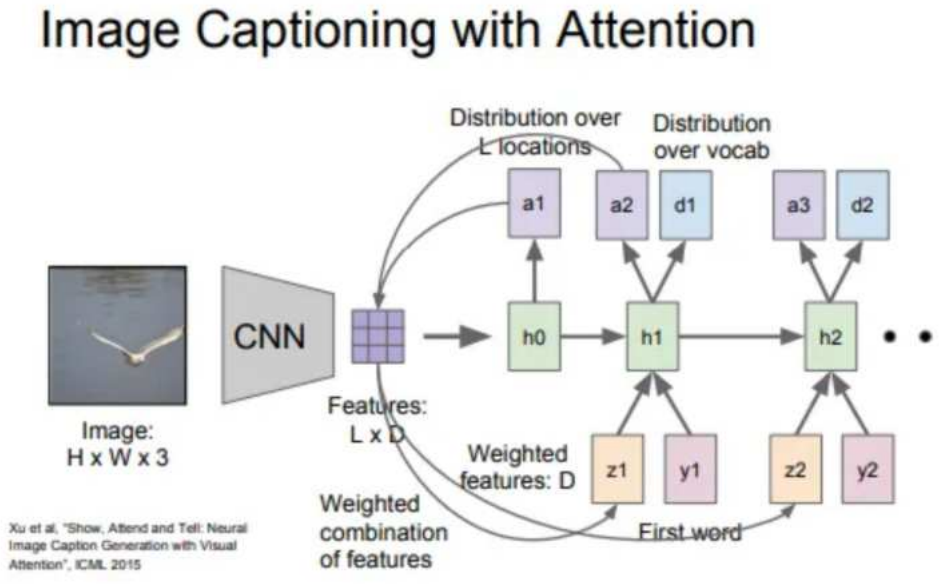
- 이 입력에서 나온 출력 y0를 input으로 다시 넣는다.
- 앞 과정을 END가 나올 때 까지 반복한다.
- 이 방법을 Top-down approach라고 한다.

10_Image Captioning with Attention

- 앞선 Top-down approach는 이미지의 디테일한 부분들에 집중하는 것이 상대적으로 어렵다는 단점을 가진다.
- Attention 기법은 이미지의 모든 부분으로 부터 단어를 뽑아내어 디테일에 신경을 써줄 수 있다.
- 이 방법을 Bottom-up Approach라고 한다.



- 이전 기법에서 하나의 벡터를 만드는 것과 다르게 각 벡터가 공간정보를 가지고 있는 grid vector(LxD)를 만들어낸다.
- grid vector를 입력으로 넣어 a1값을 생성한다
- 이 a1(attention) (L x 1)에 grid vector의 각 depth(1xD)를 곱하여 scaling을 한다.
- L개의 scaling 된 vector를 더하여 z1(1 x D)를 만들고 이를 입력으로 사용한다.



- z1와 단어를 입력으로 넣어 a2값과 단어의 분포도 값을 얻는다.

- 위에서 얻은 a2값을 다시 grid vector와 곱하여 z값을 얻고 이를 반복한다.

11_Multi layer RNNs

Multilayer RNNs

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

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

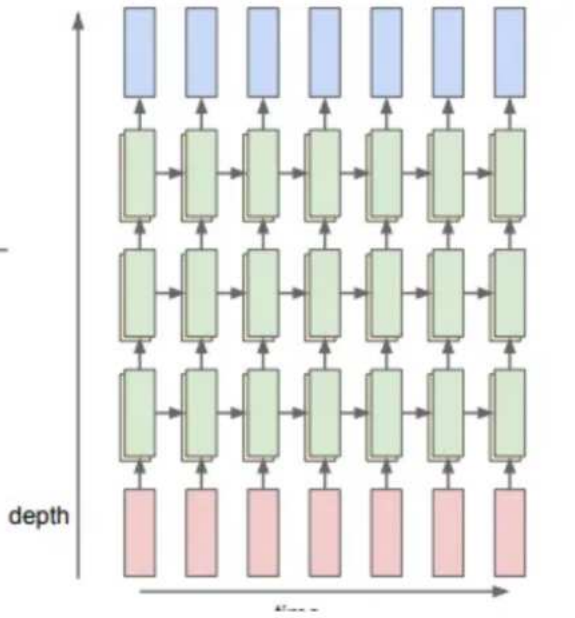
LSTM:

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

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

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

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



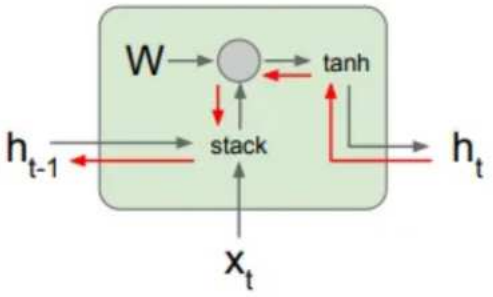
- CNN에서 layer들을 깊게 쌓았듯이 RNN도 layer들을 깊게 쌓으면 성능이 개선된다.
- 하지만 너무 깊게 쌓으면 오히려 성능이 저하될 수 있다.

12_Vanilla RNN Gradient Flow

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



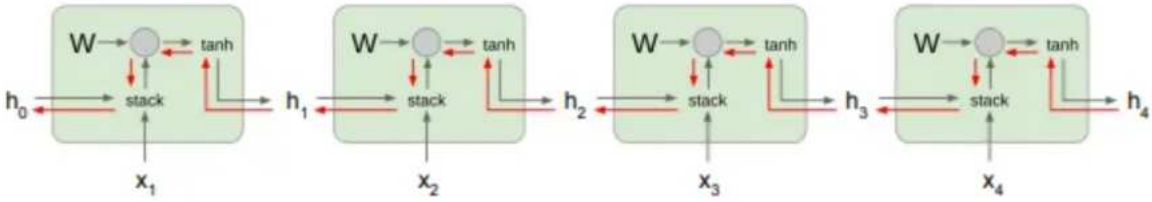
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

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

- vector를 stack해서 합칠 수 있다.

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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**

- tanh와 W를 계속 통과하기 때문에 마지막 h0의 backward는 W의 transpose 가 계속 곱해지는 형태일 것이다.
- 만약 이 값이 1보다 크면 Exploding gradient가 발생하고 1보다 작으면 vanishing gradient문제가 발생한다.
- exploding 문제는 `gradient clipping` 으로 scale을 조정해준다.
- vanishing 문제는 LSTM으로 해결할 수 있다.

13_Long Short Term Memory(LSTM)

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

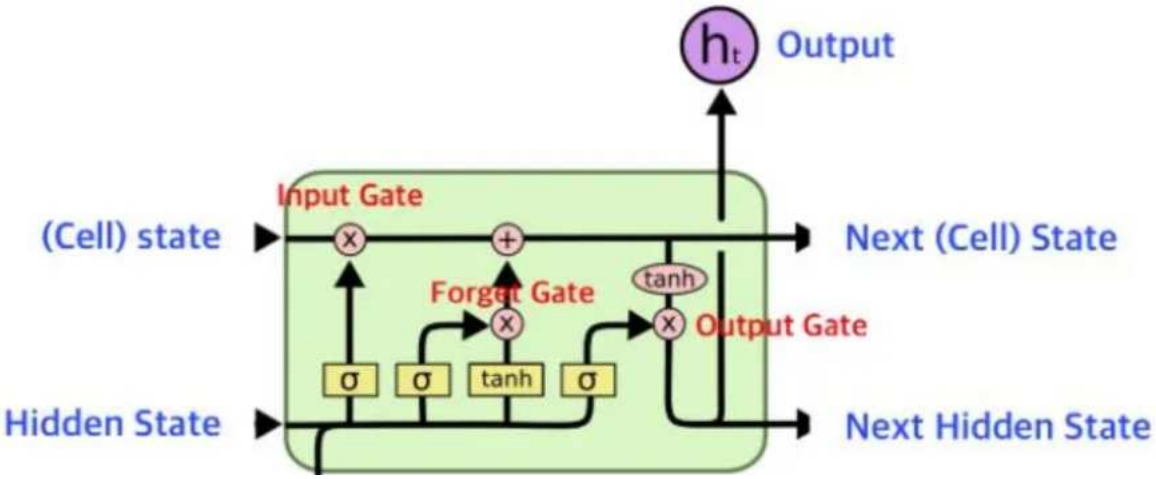
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

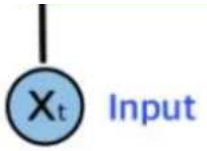
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation

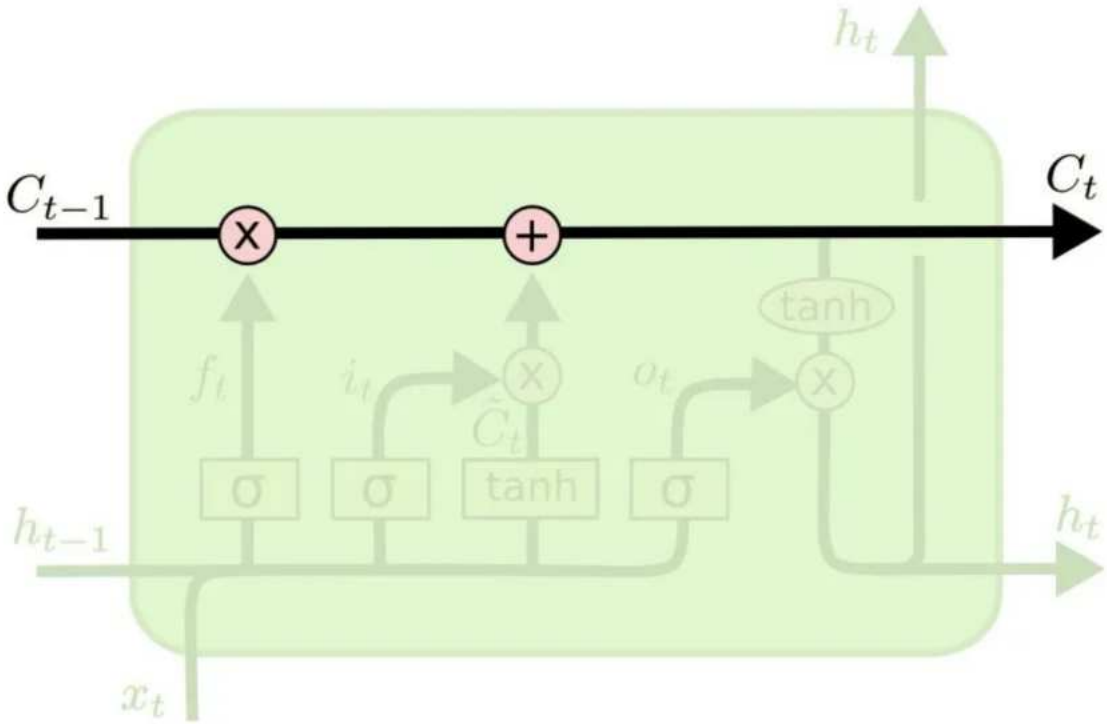
- LSTM은 한 cell당 2개의 hidden state가 있다. 하나는 h_t 이고 다른 하나는 c_t(cell state)이다.
- Cell state는 LSTM 내부에만 존재한다.
- 2개의 입력을 받아서 4개의 gates[i, f, o, g]를 계산한다.
- 그 후 cell_states를 업데이트 한 후 c_t를 사용하여 hidden_states를 업데이트 한다.



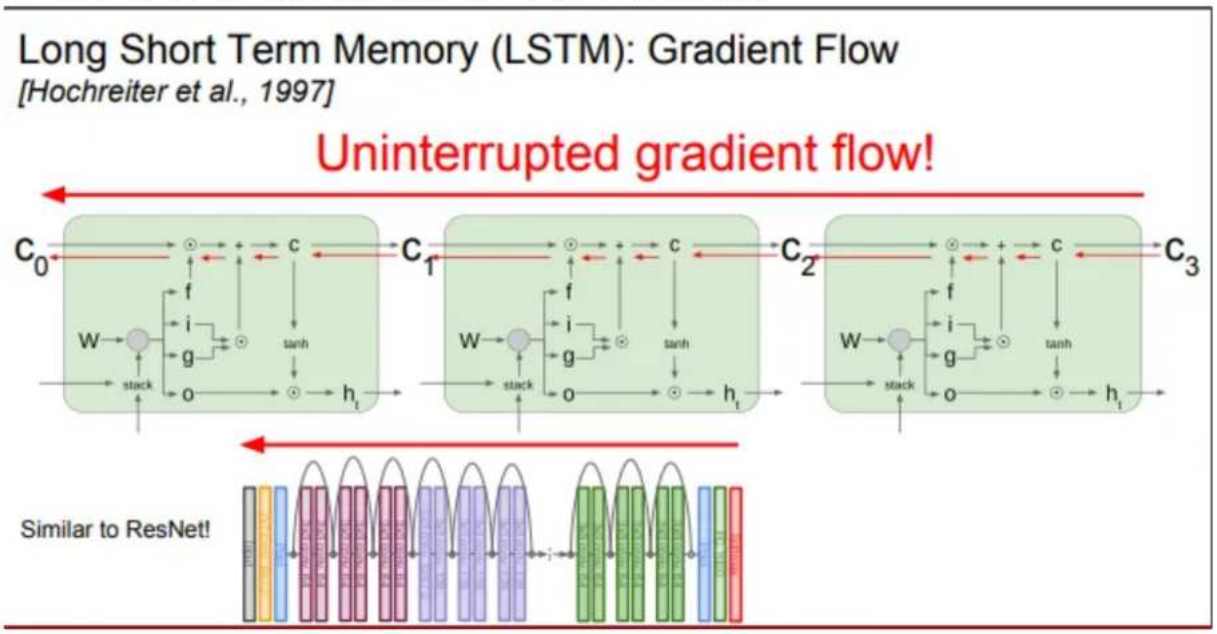


- 6개의 파라미터와 4개의 gate로 이루어져 있다.

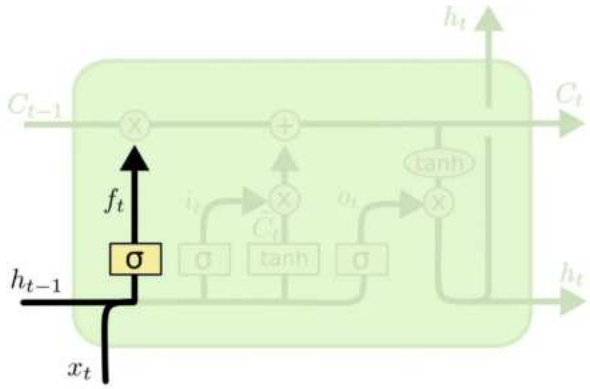
1. Cell State



- Lineara interaction만 작동시키기 때문에 gradient가 잘 작동한다.
- Gate라는 불리는 구조에 의해서 정보가 추가되거나 제거되며, Gate는 Training을 통해서 어떤 정보를 유지하고 버릴지 학습합니다.



2. Forget Gate



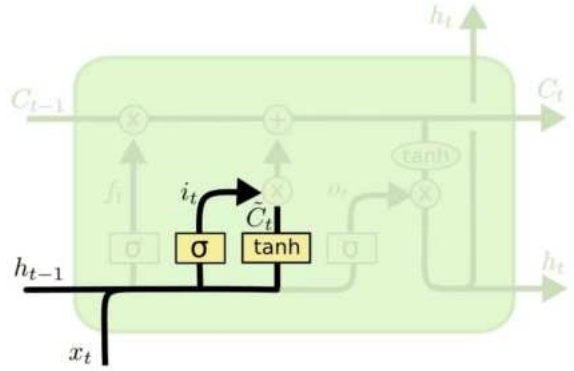
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- 과거의 정보를 버릴지 말지를 결정하는 과정이다.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- hidden과 x값을 받아서 c(t-1)에 보내줍니다. 그 값이 1이면 모든 정보를 보존하고 0미연 죄다 갖다 버립니다.

3. Input Gate

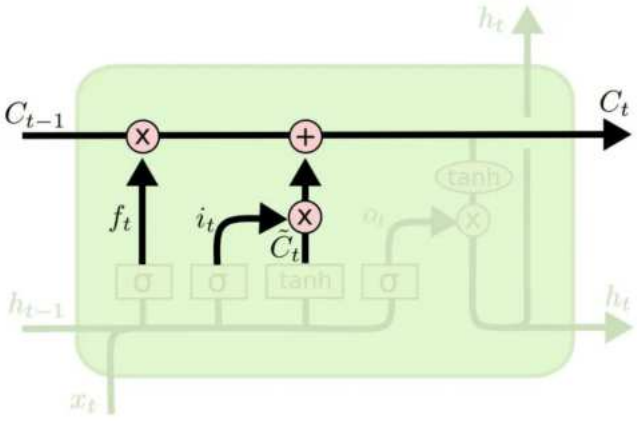


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- 현재 정보를 기억하기 위한 게이트 입니다. 현재의 Cell state값에 얼마나 더할지 말지를 정하는 역할 입니다.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

4. Update

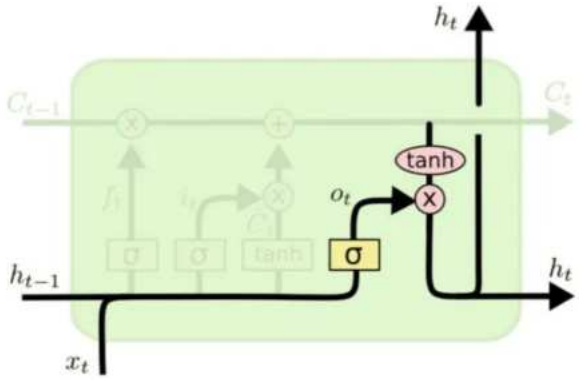


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- 이전 Cell state를 새로운 State로 업데이트 하는 과정이다. Forget Gate를 통해서 얼마나 버릴지, Input Gate에서 얼마나 더할지를 정했으므로 이 Update 과정에서 계산을 해서 Cell State로 업데이트 해줍니다.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

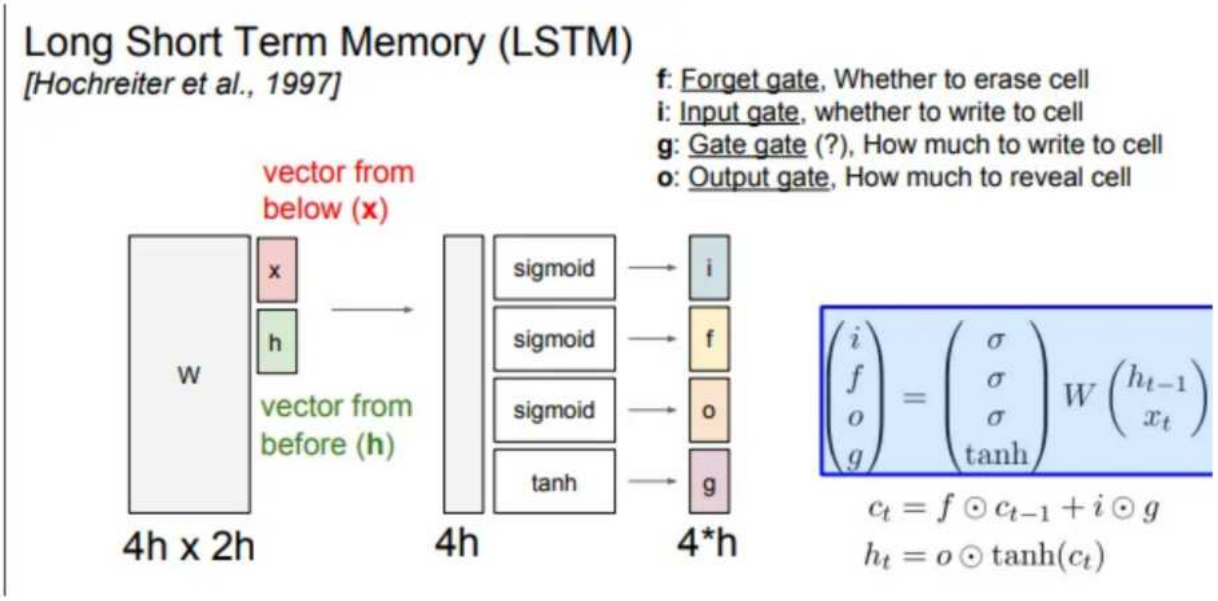
5. Output Gate



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- 어떤 출력값을 출력할지 결정하는 과정으로 최종적으로 얻어진 Cell State 값을 얼마나 뺴낼지 결정하는 역할을 한다.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$



- 다음 모든 식을 합쳐서 행렬로 깔끔하게 표현해보자.

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

- `sigmoid` : output 0~1
- `tanh` : output -1~1