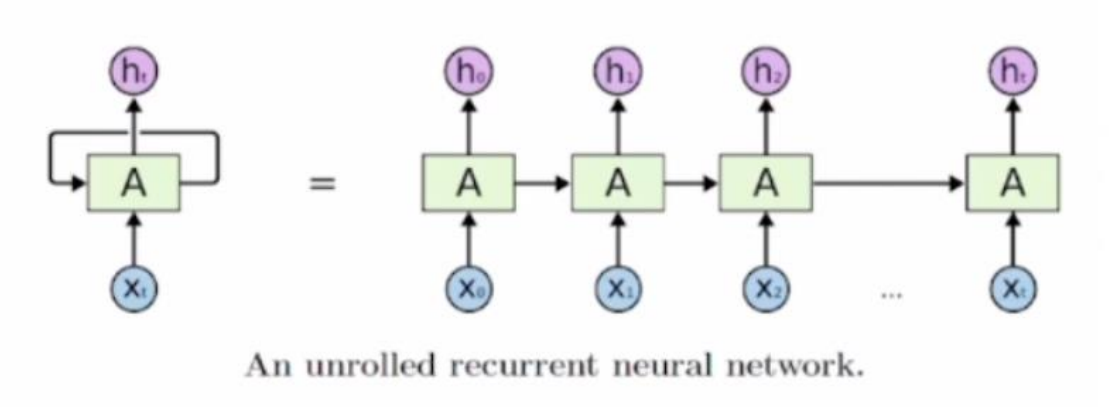
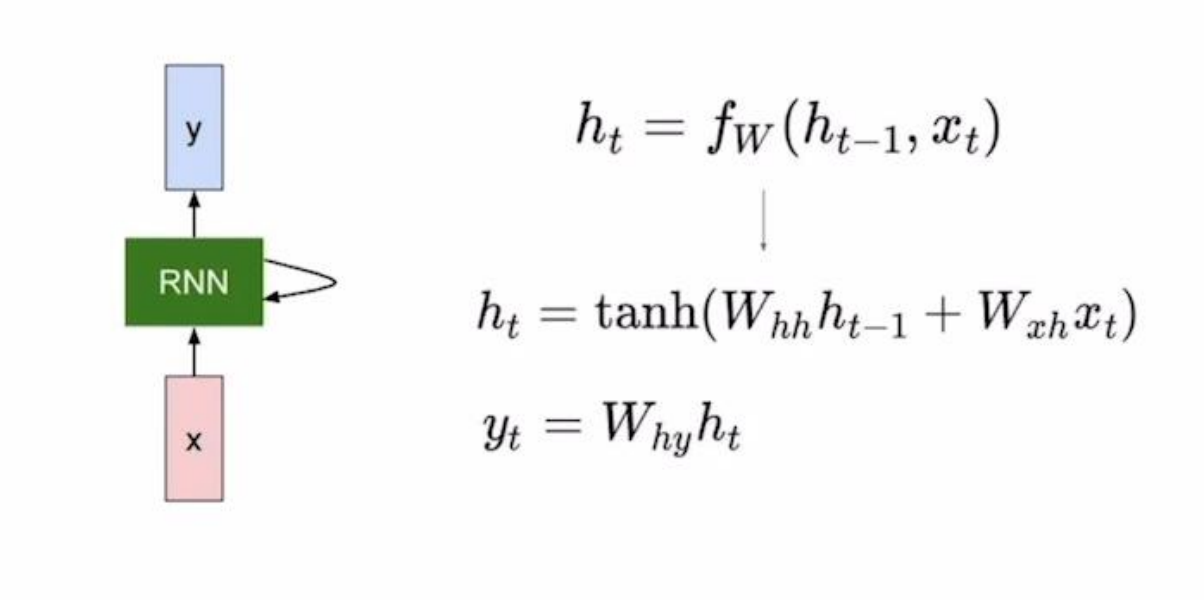
RNN

RNN(Recurrent Neural Network)



An unrolled recurrent neural network.

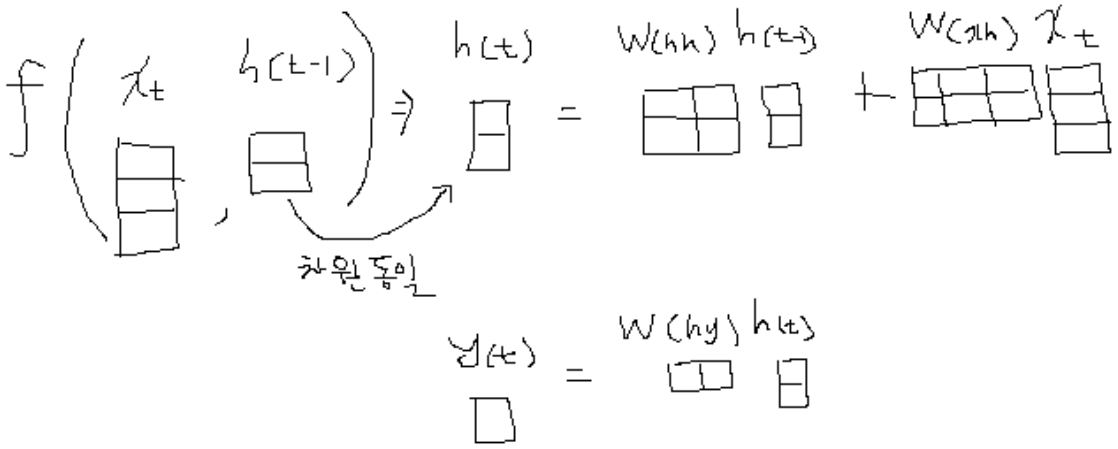
RNN 은 입력과 출력을 시퀀스 단위로 처리하는 모델이다. 현재 입력 값에 대하여 이전 입력 값들에 대한 정보를 바탕으로 예측 값을 산출한다는 말이다. 예를 들어, "I am a student"라는 문장이 있고 RNN 모델이 있다고 가정해보자. 현재 RNN 모델 안에는 무작위로 초기화된 Hidden state 가 있다. 이때 RNN 모델 안으로 처음에 "I"라는 단어가 들어가 Hidden state 를 update 하고 output(필요하다면)을 내놓게 된다. 그 다음 "am"이라는 단어가 들어갈 때에 "I"에서 만들어진 Hidden state 를 이용하여 parameter 들을 update 하고 output(필요하다면)을 내놓게 된다. 그러면 현재 hidden state 에는 "I"와 "am"의 정보가 담겨있게 되고 이런 식으로 순서가 있는 sequence 정보를 처리하는 모델을 RNN 이라고 한다.



지금부터 RNN 을 좀 더 자세하게 들여다보자.

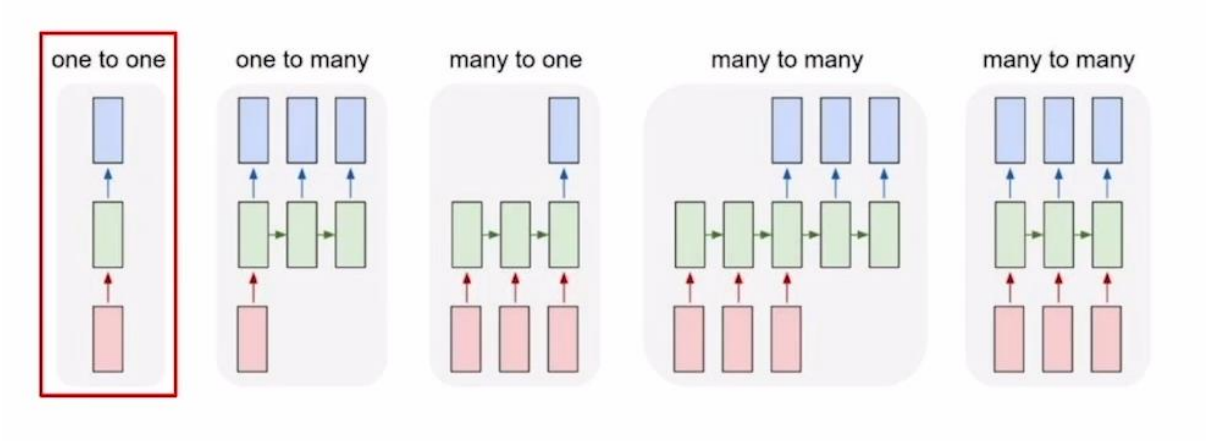
- t - 시점("I am a student"의 경우 I 는 1, am 은 2...)

- h(t-1) - t-1 시점의 hidden state
- h(t) - t 시점의 hidden state
- x(t) - t 시점의 input 값
- F(w) - RNN 함수(with parameter W)
- y(t) - t 시점의 output 값
- tanh() - activation function tanh



앞에서 RNN 은 이전시점까지의 Hidden state 를 사용한다고 말했다. 그래서 RNN 의 입력으로는 이전 시점까지의 hidden state(h(t-1))와 현재시점의 input 값(x(t))이 들어가게 된다. 두 값이 RNN(F(W)) 모델에 input 으로 들어가 선형변환을 거치게 되고 tanh 함수를 이용하여 현재시점의 Hidden state(h(t))를 생성하게 된다. 그 후 해당 t 시점의 output 이 필요하면 h(t)에 또 한번의 선형변환을 통하여 y(t)를 구하게 된다.

Types of RNN



앞에서 RNN 에 대하여 설명할 때, 매 시점마다 Output 을 구하는 것이 아니라 필요하다면 Output 을 구한다고 계속 말했다. RNN 에서는 이러한 Output 의 구조에 따라 여러가지 type 이 존재하게 된다.

one-to-one(standard neural net)

[키, 몸무게, 나이]와 같은 정보를 입력 값으로 할 때, 이를 통해 저혈압/고혈압인지 분류하는 형태의 태스크가 이 경우에 해당한다.

one-to-many(image captioning)

하나의 이미지를 입력으로 주면 이미지에 대한 설명 글을 시점별로 생성하는 형태의 task 가 여기에 해당한다. 원래 앞에서 말하기로는 RNN 은 매 시점마다 input 을 준다고 말했다. 하지만 이 경우 입력이 처음 step 에만 들어가는 것처럼 보인다. 그래서 이 경우 첫 시점 외에 나머지 시점에는 O 으로 이루어진 Tensor 를 입력으로 주게 된다.

many-to-one(sentiment classification)

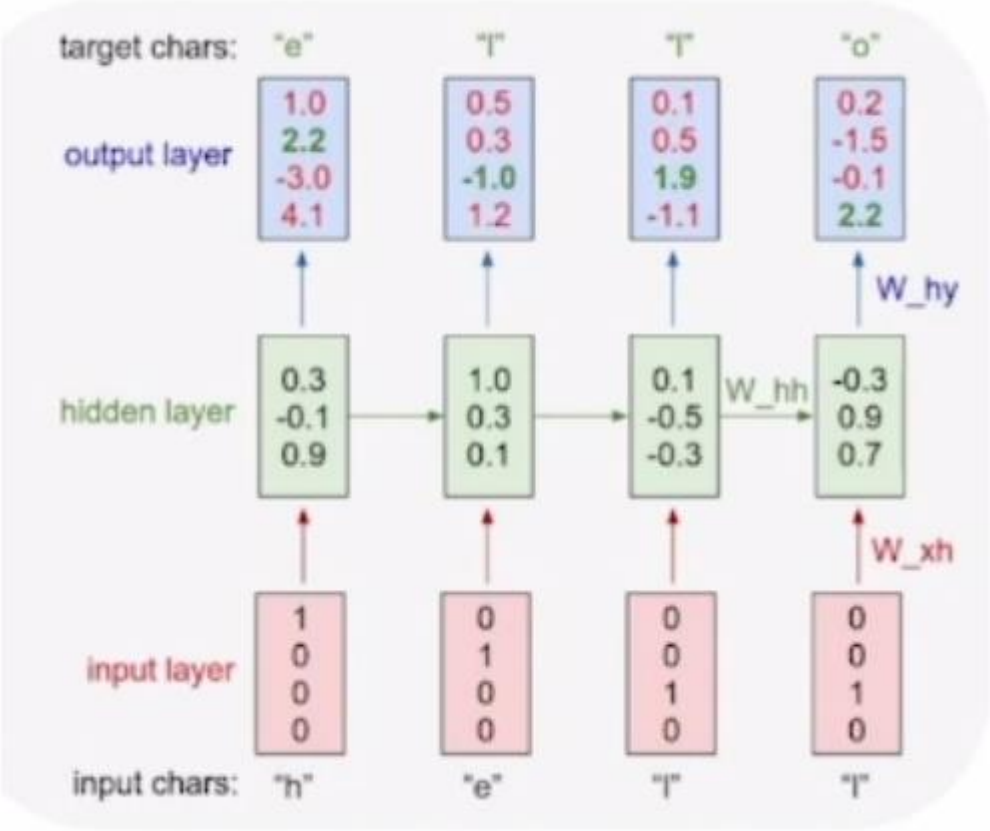
하나의 문장을 sequence 순서대로 입력을 주고 마지막 시점에 나온 hidden state 를 가지고 output 을 구하여 하나의 결과를 내는 경우가 여기에 속하게 된다.

many-to-many(machine translation)

i go home 과 같은 문장을 입력으로 주고, 나는 집에 간다 라는 문장을 output 으로 뽑아주는 이러한 task 가 이 경우에 해당한다. 이 경우는 입력을 줄 때 마다 output 을 구해줘야 한다.

Character level language model

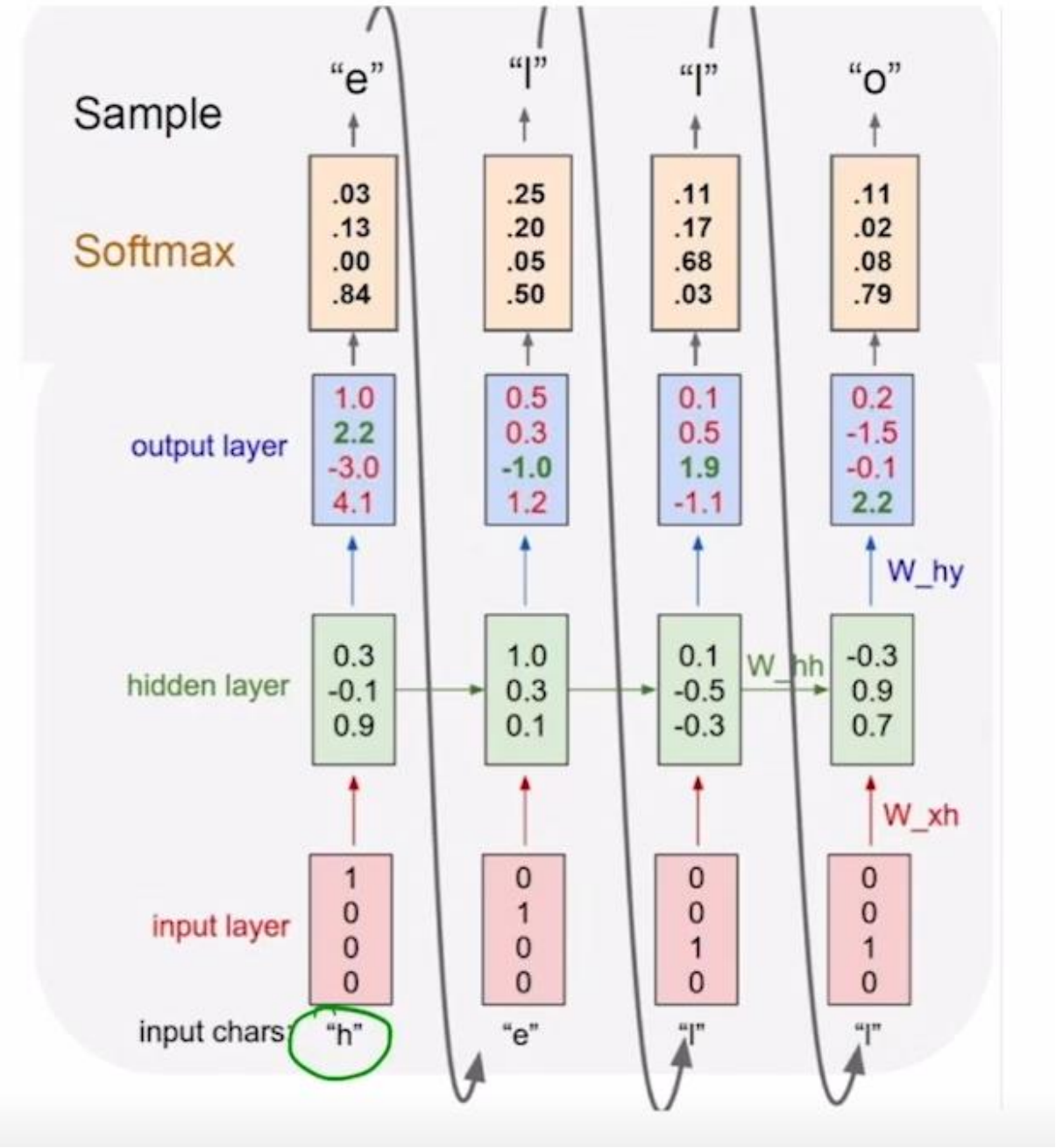
training



- many-to-many type 으로 Character level 에서 다음 Character 를 예측하는 언어모델
- hello 라는 단어가 있을 때 h 가 주어지면 e 를 예측하는 방식으로 hidden state 학습
- 다음에 올 단어를 예측하는 task 이기 때문에 마지막 output 의 dimension 은 총 단어의 수
- 마지막 output logit 에 softmax 함수를 통하여 각 단어일 확률을 구하고 loss 를 전달

  예시를 보면 3, 4 번째 input 이 동일하지만 결과는 l, o 로 달라져야 한다 -> hidden state 의 차이

inference



- 학습을 마친 후 inference 를 진행할 때에는 이전의 output 값을 다음 step 의 input 으로 사용

language model 의 방법은 character level 보다 더 높은 word level 차원에서도 학습 가능하고 실제 코드쓰기나 소설쓰기 등 다양한 task 가 가능함

다양한 언어모델의 예시

```
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

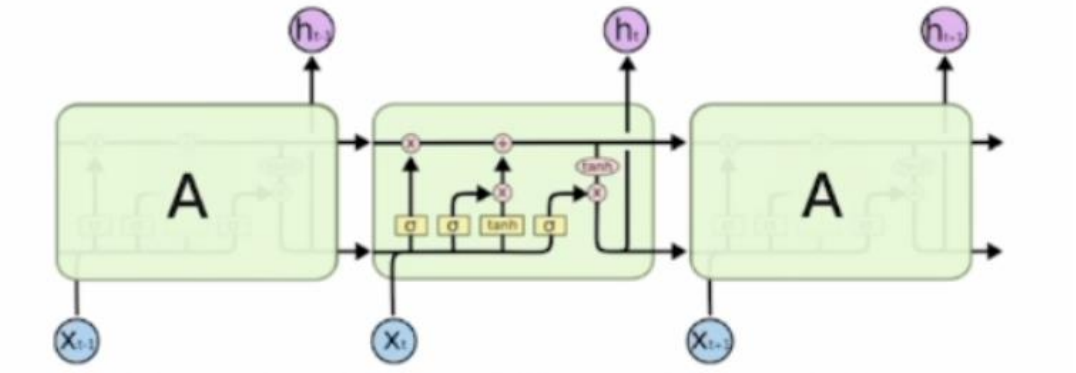
```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

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.
```

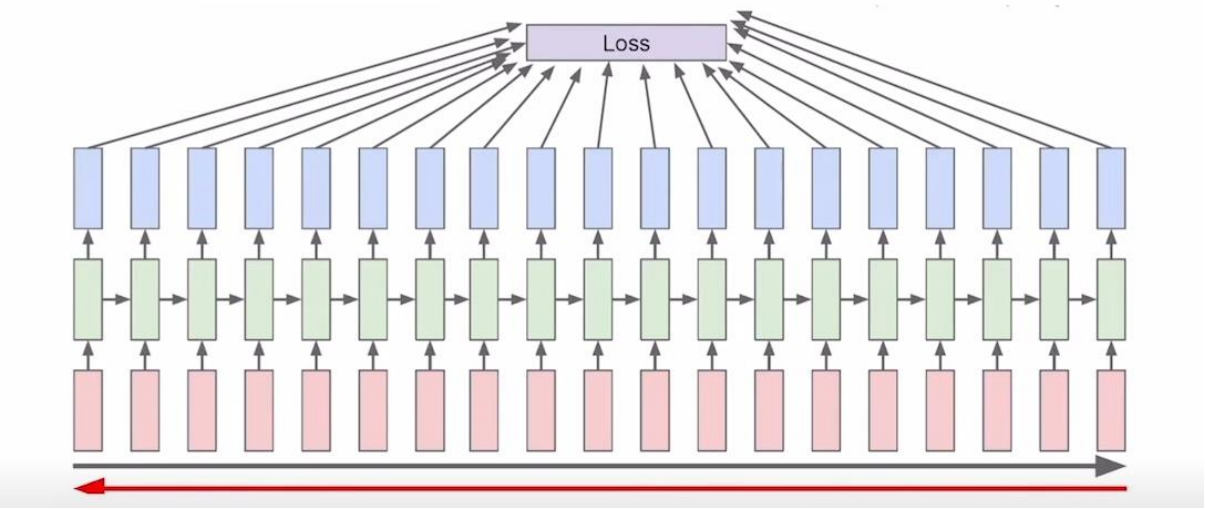
train more

```
"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.
```



The repeating module in an LSTM contains four interacting layers.

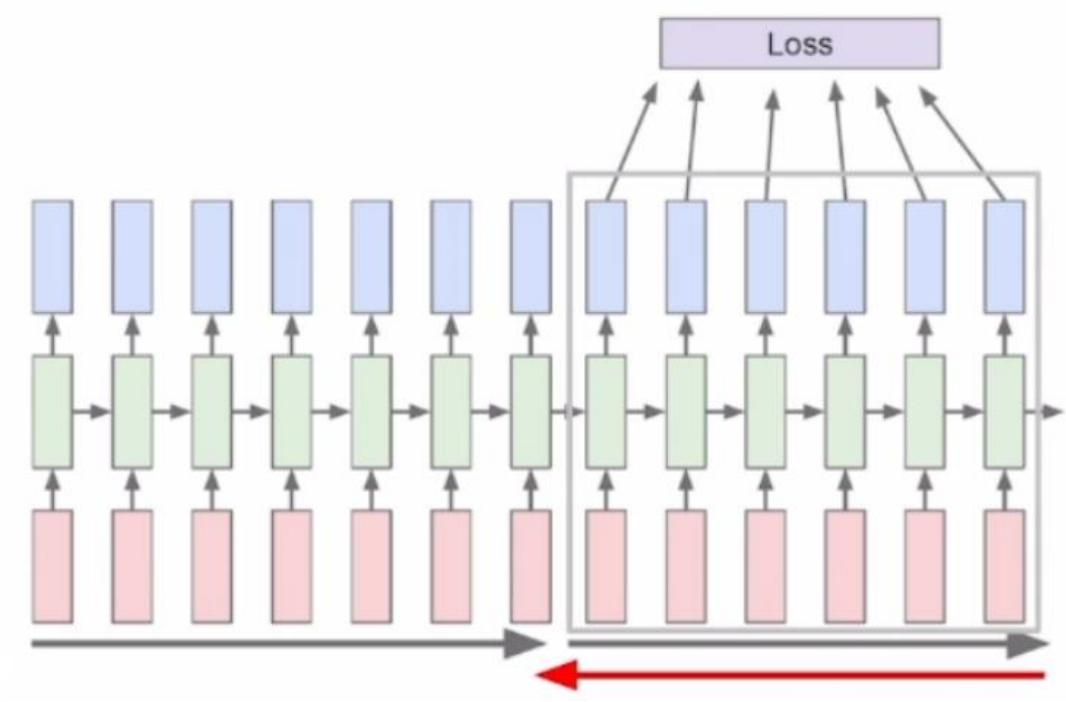
지금까지는 RNN 과 RNN 을 이용한 CHARACTER LEVEL language model 에 대해서 살펴보았다.



CHARACTER LEVEL language model 은 위의 그림과 같이 학습이 진행된다. 각각의 input 에 대하여 이전의 hidden state 와 선형결합으로 현시점의 hidden state 를 구하고 각각의 시점에서 해당하는 hidden state 에 선형결합으로 output logits 를 구하게 된다. 그 후 softmax 함수를 통과시키고 예측된 label 값에 loss 를 계산하고 backpropagation 를 통하여 각각의 가중치들을 update 하며 학습을 하게 된다

Truncated Backpropagation

RNN 을 학습을 시킬 때에 마지막에 backpropagation 을 통하여 hidden state 를 update 한다고 위에서 설명했다. 이 backpropagation 을 하는 과정에서 입력 sequence 의 길이가 너무 길다면 한번에 backpropagation 을 하는 것은 resource 적인 측면에서 비효율적일 것이다.

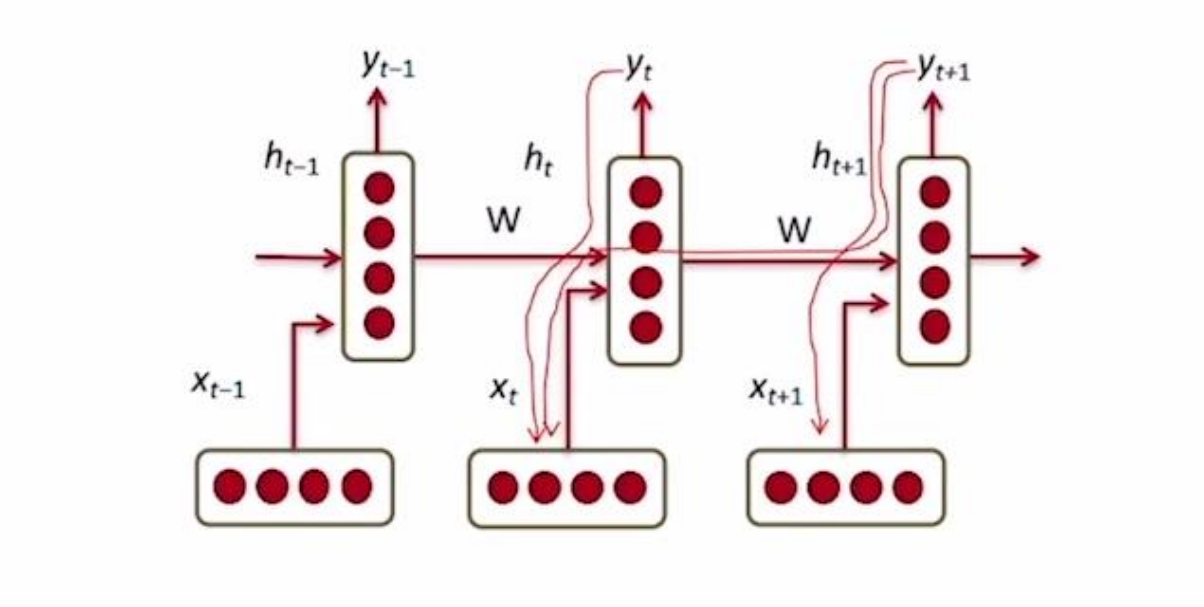


따라서 일정 sequence 길이로 잘라서 나눠서 backpropagation 을 수행하게 된다.

Role of Hidden State

RNN 은 이전 step 까지의 정보를 hidden state 에 저장한다. 위의 그림은 hidden state 의 숫자를 음수면 빨간색 양수면 파란색으로 시각화 한 것인데 이 hidden state 가 따옴표를 열고 닫았던 정보를 기억하고 있는 것을 볼 수 있다.

Vanishing/Exploding Gradient



RNN 은 위에서 말한 형태로 학습이 진행되지만 몇가지 문제점이 존재한다. RNN 의 경우 다음 step 으로 넘어가는 데에 이전의 hidden state 에 가중치 W(hh)를 곱하고 input 값에 W(xh)를 곱하는 연산을 반복적으로 진행하게 되고 원하는 정보를 가진 hidden state 를 얻기 위해 backpropagation 을 수행하게 된다.

Toy Example

- $h_t = \tanh(w_{xh}x_t + w_{hh}h_{t-1} + b), t = 1,2,3$
- For $w_{hh} = 3, w_{xh} = 2, b = 1$

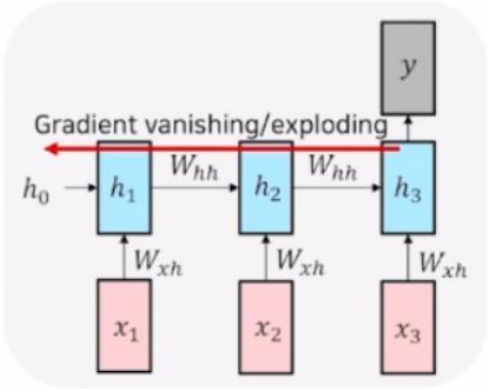
$h_3 = \tanh(2x_3 + 3h_2 + 1)$

$h_2 = \tanh(2x_2 + 3h_1 + 1))$

$h_1 = \tanh(2x_1 + 3h_0 + 1))$

...

$h_3 = \tanh(2x_3 + 3\tanh(2x_2 + 3h_1 + 1) + 1)$



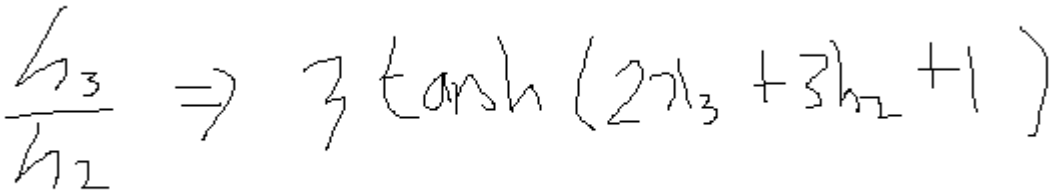
위 그림을 통해 time step 이 3 인 RNN 의 backpropagation 과정을 살펴보자. 3 번째 step 에서 결과를 가지고 backpropagation 을 통하여 오차를 1 번째 step 까지 전달 하기 때문에 h1 에 대하여 h3 를 미분하게 되면 chain rule 에 의해서 아래와 같은 식이 만들어 진다.(미분기호 생략)

$$\frac{h_3}{h_1} = \frac{h_3}{h_2} \cdot \frac{h_2}{h_1}$$

먼저 h3 에 대해 h2 의 미분을 먼저 해보자.

$$f(ax+b) \Rightarrow af'(ax+b)$$

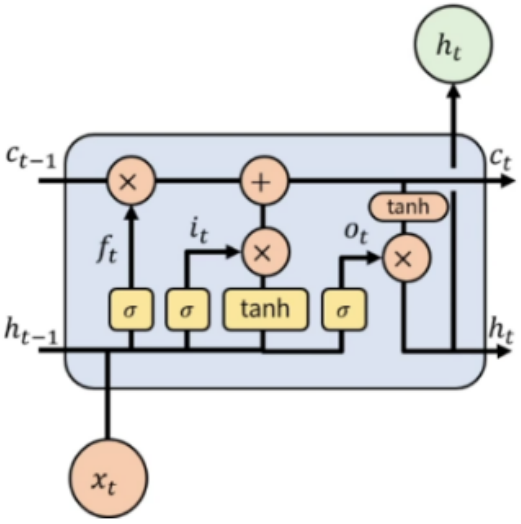
위와 같은 tanh 의 속미분 공식으로 인해 3 이 앞으로 나오게 되고 아래와 같은 결과가 나오게 된다.



이 과정을 h2 에 대해 h1 의 미분에 또 적용을 시킨다면 속미분에 의해 3 이라는 값이 나오게 될 것이다. 만약 이게 3 step 이 아니라 100 step 이었다면 3 의 100 승이라는 결과가 나오게 될 것이다. 이러한 문제를 exploding gradient 라고 한다. 그리고 만약 3 이 아니라 0.2 와 같은 분수의 꼴이었다면 곱하면 곱할수록 0 에 수렴했을 것이다. 이를 vanishing gradient 라고 하고 해당 상황(vanishing gradient)에 처한다면 학습이 잘 되지 않을 것이다.

LSTM

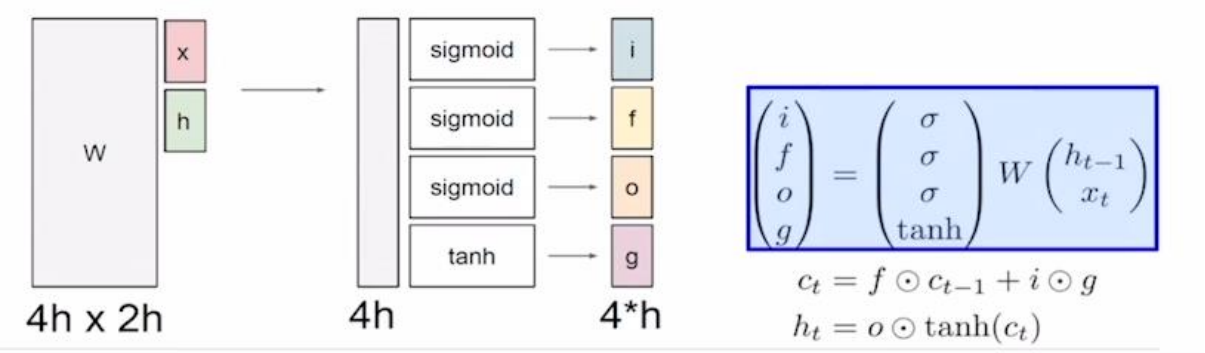
LSTM 은 기존의 RNN 에 존재하던 vanishing/exploding gradient 의 문제 해결하고 Time step 이 멀더라도 효율적으로 정보를 전달하게 하는 모델이다.



$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

LSTM 은 기존의 RNN 과 달리 현시점의 input(x(t))와 이전 시점의 hidden state(h(t-1)), 이전 시점의 cell state(c(t-1))를 입력으로 받는다.

여기서 cell state 벡터는 무엇일까? cell state 벡터는 hidden state 에서 한번의 과정을 더 거쳐서 그 time step 에서 필요한 정보만을 남기는 방식으로 만들어지게 된다. LSTM 의 연산과정을 더 자세하게 살펴보자.



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

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

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

LSTM 에는 총 4 개의 gate 가 존재한다.

- f - forget gate(whether to erase cell)
- i - input gate(whether to write cell)
- o - output gate(how much to reveal cell)
- g - gate gate(how much to write cell)

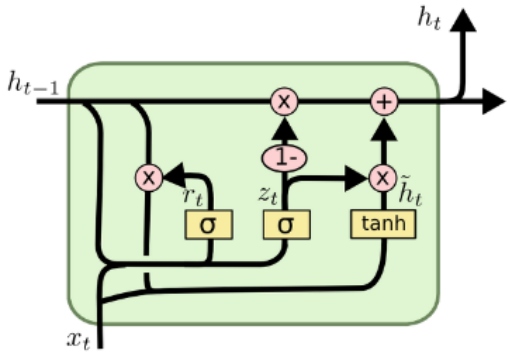
input, forget, output, gate gate 는 각각 입력으로 들어온 x(t)와 h(t-1)을 가지고 선형변환을 수행 후 활성화 함수를 거쳐 각각의 정보를 가지게 된다. 그 후 구해진 개별 gate 값들을 가지고 c(t)와 h(t)를 구하게 된다.

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

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

c(t)는 이전 시전의 c(t-1)과 앞에서 구해진 forget gate 의 곱과 input gate 와 gate gate 의 곱의 합으로 구해지게 되고 h(t)는 이렇게 구해진 c(t)에 tanh 함수를 거친 후 output gate 와의 곱으로 구해지게 된다. 따라서 backpropagation 진행시 가중치(W)를 계속해서 곱해주는 연산이 아니라 forget gate 를 거친 값에 대해 필요로 하는 정보를 덧셈을 통해 연산하여 vanishing/exploding gradient 문제를 방지하게 된다.

GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU 는 LSTM 을 간략화 해서 계산을 좀 편하게 해주는 모델이다. 기존의 LSTM 과의 차이점은

LSTM 은 입력 값으로 h(t-1), c(t-1), x(t)를 받지만 GRU 는 h(t-1), x(t) 두개만 입력으로 주어진다는 점이다. GRU 의 구조는 위와 같으며 LSTM 과는 달리 cell state 가 완전히 사라졌으며(hidden state 와 하나로 합쳐짐) gate 도 forget, input 2 개만 있는 것을 알 수 있다.