Sequence Models

한 단어를 표시하는 X<1> - 일시적인 시퀀스

시퀀스의 길이 : T_X, T_y

X(i) : i 훈련

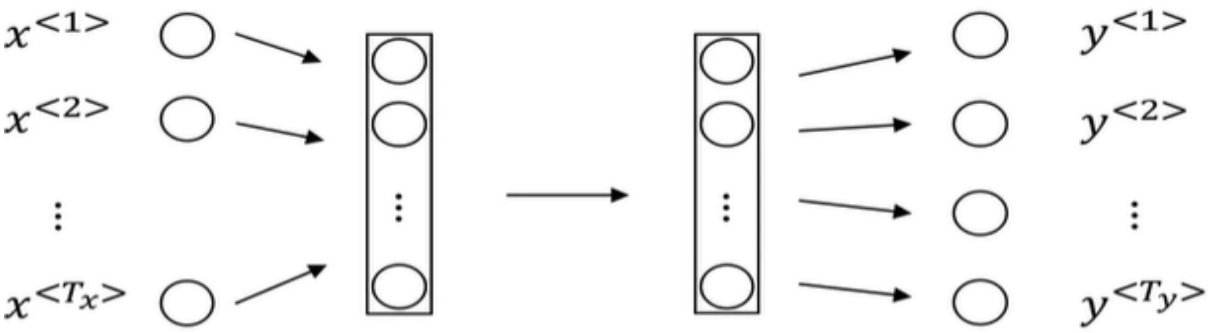
X(i)<t> : T_X 가 시퀀스의 길이라면, 훈련 세트 내 다른 예시의 길이가 다를 수 있다.

T_X(i) : 훈련 예시 i 에 대한 입력 시퀀스 길이

단어 사전 10000 개 ->단어 하나는 원핫벡터로 단어사전의 i 번째에 있는 10000 차원 벡터가 된다

* 단어사전에 없는 단어이면, UNK(고유 토큰)로 표시되는 모르는 단어라 불리는 가짜단어 만듦

[X -> y 로 매핑]

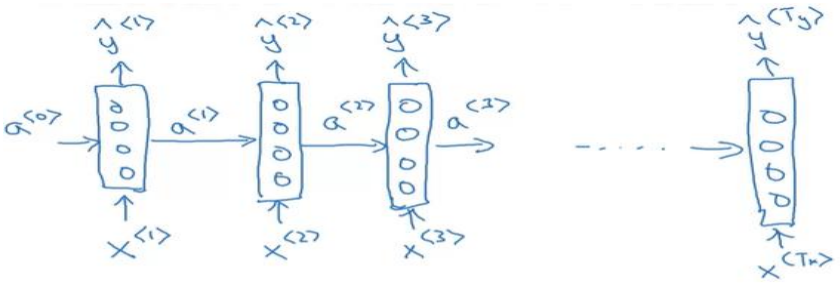


문제점)

- 입력과 출력은 다른 길이와 다른 예시가 될 수 있다. 같은 입력 길이 T_X 이거나 같은 대문자 길이 T_y 를 가지지 않고, 모든 문장은 임의의 값 0 을 채워 넣을 수 있어 최대길이가 아니다.
- 나이브 신경망 아키텍처는 텍스트의 서로 다른 위치에서 학습한 기능을 공유하지 않는다.

모델의 매개변수 수를 줄일 수 있다. 이전에 X<t> 각각은 10000 차원 원핫벡터이며,
입력 크기가 최대 단어 수 * 10000 인 아주 큰 입력 레이어이다.
첫 번째 계층의 가중치 행렬은 많은 매개변수를 가진다. -> 순환신경망은 이러한 단점이 없다.

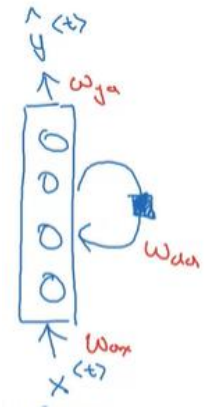
[순환신경망] many – to – many



가짜 활성화 a<0> : 영벡터 무작위로 초기화

T_X = T_y

좌에서 우로 문장을 읽으면, 처음 읽는 단어를 X<1>로 가정하고, 이를 신경망 레이어에 넣는다. 순환신경망이 하는 것은 문장의 두 번째 단어 X<2> 읽을 때, y 햇<2>예측할 때 X<2>만 사용하는 것이 아니라, 첫 번째 단계에서 연산한 정보의 일부도 가져온다. (첫 번째 단계의 비활성화 값이 두 번째 단계에 전달)



순환 연결을 표시하기 위해 레이어가 셀에 피드백하는 루프 그램
음영 처리된 상자를 그려 한 단계의 지연 시간 나타냄

입력 단어에서 히든 레이어로 연결을 관리하는 매개변수 세트 W_ax 는 각 단계에서 사용하는 매개변수와 동일

비활성화, 수평연결은 모든 시간 단계에서 사용되는 일부 매개변수 세트 W_aa

출력 예측을 제어하는 W_ya

y 햇<3>의 예측은 X<1>, X<2>, X<3>의 정보를 모두 사용하는데, RNN 의 약점은 앞서 나온 정보만을 사용해 예측한다는 것이다. 즉, X<4>이후의 정보는 사용하지 않는다. => 문제 !

ex1) He said, "Teddy Roosevelt was a great President."

Teddy 라는 단어가 사람의 일부인지 결정하기 위해 첫 두 단어의 정보뿐만 아니라 문장 중 그 뒤에 있는 단어들의 정보를 아는 것도 더 유용하다. 첫 두 단어(He, said)만 고려하면, Teddy 가 사람의 이름의 일부인지 확실히 알 수 없다.

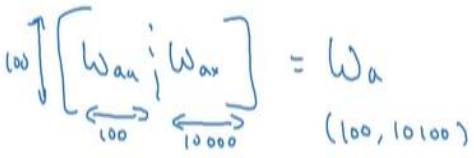
a<1> = g(W_aa * a<0> + W_ax * X<1> + b_a) <- tanh, Relu

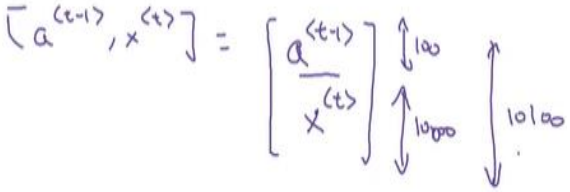
y 햇<1> = g*W_ya * a<1> + by) <- sigmoid, softmax

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

$$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$$





W_a : W_aa 와 W_ax 를 수평적으로 쌓아 놓음
a 가 100 차원, X 가 10000 차원이면,
W_aa : 100 * 100 차원 행렬,
W_ax : 100 * 10000 차원 행렬
W_a : 100 * 10100 차원 행렬
[a<t-1>, X<t>] : 10100 * 100 차원 행렬

$$a^{\langle t \rangle} = g(W_a[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_a)$$

[역전파] Backpropagation through time

<요소별 손실> : 한 단어에서 단일 위치 또는 단일 시간 세트에서의 단일 예측값과 연관된 손실

$$\mathcal{L}^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) = -y^{\langle t \rangle} \log \hat{y}^{\langle t \rangle} - (1-y^{\langle t \rangle}) \log (1-\hat{y}^{\langle t \rangle})$$

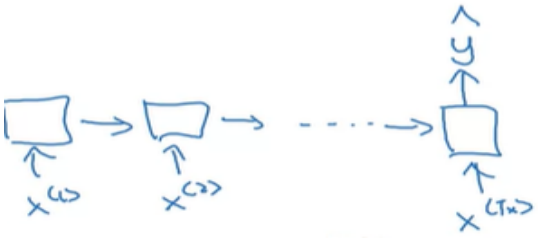
손실함수에서 y 햇<t> : 사람의 이름일 확률

y<t> : 사람 이름이면 1, 아니면 0

<전체 시퀀스의 전체 손실> : 시간 1 부터 T_y(= T_X)까지 요소별 손실의 합

[RNN 아키텍처]

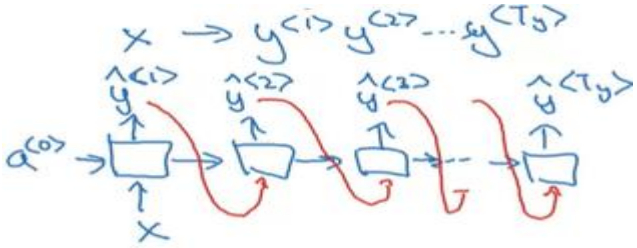
- many – to – one



매 순간 출력을 사용해야 하는 것이 아니라 이미 전체 문장이 입력되었다면 RNN 이 전체 문장으로 읽어들여 마지막 단계에서 y 햇을 출력 가능, 많은 단어를 입력한 다음 단 하나의 숫자 출력

ex. 감성분석(긍/부정)

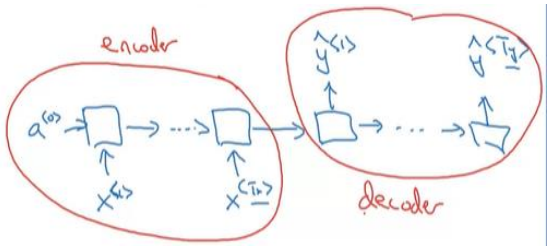
- one – to – many



입력이 한 개거나 없을 수도 있다. 합성된 첫 번째 출력값 y 햇<1>을 다음 레이어에 전달

ex. 음악 생성

- many – to – many



입력과 출력 길이가 다른 경우(T_X != T_y)

ex. 번역

프랑스어 문장인 입력을 받아들이는 인코더

문장을 읽고 다른 언어로 번역을 출력하는 디코더

[언어 모델]

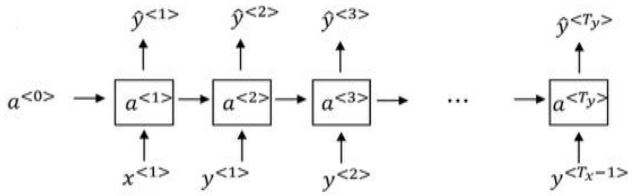
어떤 문장이 주어지든 그 특정한 문장의 확률을 추정

RNN 사용해 언어 모델 만들려면, 텍스트의 코퍼스(분량이 큰, 수십 개의 영어 문장을 의미하는 NLP 용어)가 포함된 훈련 셋이 필요

1. 문장을 토큰화 : 단어들을 각각 원핫벡터나 어휘의 색인에 맵핑
   문장이 끝날 때를 모델하고 싶을 때, 추가 토큰을 더하는 EOS(end of sentence_
   문장이 언제 끝나는지를 알아낼 수 있도록 한다.

   EOS 토큰을 추가하려는 경우, 예로 단어가 9 개이면, 9 개의 input 이다.
   * 마침표도 토큰이어야 하는지 결정

2. 확률을 모델링하기 위해 RNN 빌드
   X<t> = y<t-1>



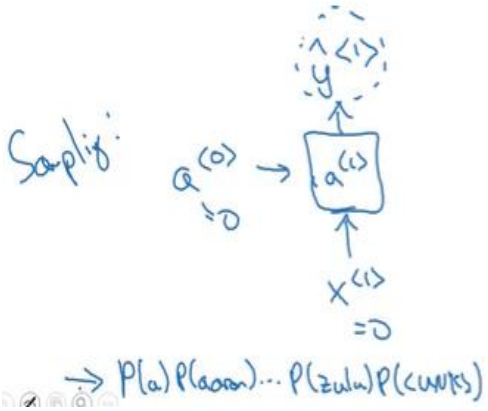
처음에는 일부 a<1>을
일부 X<1>의 함수로서 컴퓨팅

X<1>, a<0>은 영벡터로 설정

a<1>은 소프트맥스 예측을 만들어 첫 번째 단어의 확률 파악 -> 아무 단어의 확률 예측
y 햇<1>은 소프트맥스에 따라 출력 / 만약 10000 개 사전에 10002 개 어휘가 있다면,
미등록어는 하지 못하고 문장은 2 개의 추가적 토큰을 가진다.

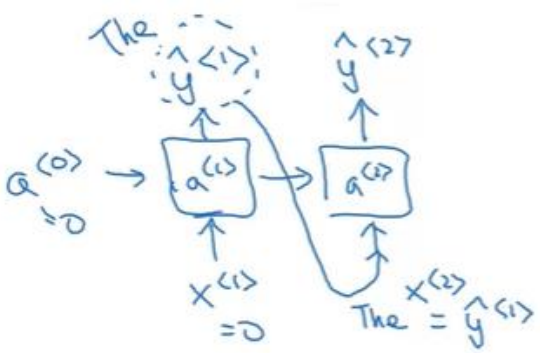
ex. 세 단어 y1, y2, y3 의 새로운 문장 한 개에서 전체 문장의 확률이 어느 정도인지
-> P(y<1>, y<2>, y<3>) = P(y<1>) * P(y<2> | y<1>) * P(y<3> | y<1>, y<2>)

[Sampling Novel Sequences] : RNN 언어모델에서 무작위로 선택된 문장을 생성하는 방법



무작위로 y 햇<1> 소프트맥스 분포에 따라 표본 매김

np.random.choice 사용하여 벡터 확률에 의해 정의된 분포에 따라 샘플링하고, 첫 번째 단어를 샘플링함



두 번째 단계에서 y<1>을 입력으로 함
+ 샘플링한 y 햇<1>을 가져다 입력으로 전달됨

P(____|the) np.random.choice 샘플링 함수 사용하여, 첫 번째 단어가 the 라는 두 번째 단어를 찾아냄

EOS 토큰 생성할 때까지 샘플링 계속할 수 있다.
= 문장의 끝을 맞았다는 것을 의미하고, 멈출 수 있다.

** Unknown 토큰을 생성하지 못하도록 하려면, UNK 토큰으로 나온 모든 표본을 거절하고, UNK 토큰이 아닌 단어가 나올 때까지 나머지 어휘와 resampling 유지

<Character-level language model>

Vocabulary = [a, b, c, ..., z, ␣, ., ,, ;, 0, ..., 9, A, ..., Z]

Ex. cat average => c : y1 / a : y2 / t : y3 / 공백 : y4

장점 : UNK 토큰에 대해 걱정할 필요 없음
단점 : 훨씬 더 긴 배열로 끝남 -> 문장의 초기 부분이 문장의 뒷부분에도 영향을 많이 줄 때 성능이 좋지 않음, 또한 계산 비용이 든다.

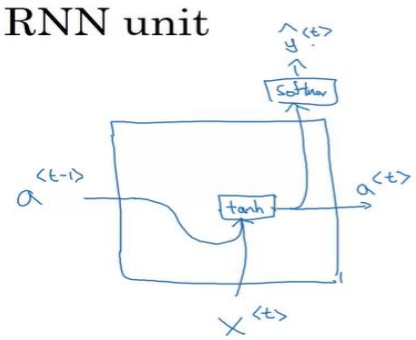
[Vanishing Gradients with RNNs]

The cat, which already ate ..., was full
The cats, which already ate ..., were full

문장의 단어가 긴 시간동안 의존성을 가지는 예이다. 문장 초반부의 단어가 후반부에 영향을 끼칠 수 있다는 것인데, RNN 은 이와 같은 장기적인 의존성을 확인하는데 효과적이지 않다. (긴 시퀀스를 처리하는 데 한계 존재)
단수/복수 명사가 문장 초반에 존재했다가 동시 사이에 which~ 같은 수식어구가 길면, 동사자리에 올 때까지 기억하고 있어야 하는데, RNN 의 경우에는 가까이에 있는 것에 영향을 더 많이 받는다. => Basic RNN 의 약점

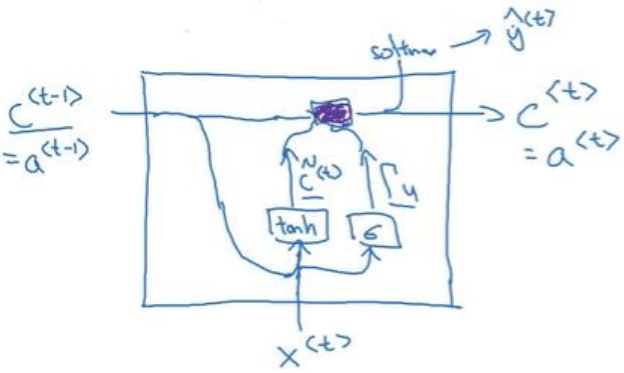
[GRU (Gated Recurrent Unit)]

장거리 의존성 잘 캡처하기 위해, 기울기 소멸 문제 해결하기 위해 RNN 의 hidden layer 수정



$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

a<t-1> : 마지막 시간 단계에 대한 비활성화



시간 t 에서 메모리 세포는 t 의 값 c 를 가짐

c<t> = a<t> (c = memory cell)

** GRU 에서 c 는 output 활성화 a 와 동일

이전 정보와 현재 입력을 통해 연산 (메모리 셀을 대체할 후보)

$$\tilde{c}^{<t>} = tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

현재 time step 에서 다음 t 로 전달할 정보들의 후보군 업데이트

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

Update gate 를 통해 어떤 정보를 업데이트할 지 결정 (0 or 1)

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

최종적으로 다음 time step 으로 전달될 c<t> 계산 (보라색 연산)

: 현재 t 에서 업데이트할 후보군과 이전 기억 정보들의 가중치 합으로 계산 / * : element-wise 곱

여기서 Gate 는 어떤 정보를 더 포함시킬지 결정하는 역할 (gate = 0 : 업데이트 하지 말고 이전 값을 고수하고 잊지 말라는 얘기)

<FULL GRU> : update gate, reset gate 존재 (h : a<t> = c<t>)

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) + c^{<t-1>}$$

Reset Gate : 이전의 정보를 적당히 리셋시키는 목적으로 시그모이드 함수를 출력으로 사용해 0~1 사이의 값을 이전 정보에 곱해주게 된다.

gamma r : c<t-1>이 c<t>의 후보를 계산하는데 얼마나 적절한지 알려줌

[LSTM (Long Short Term Memory)] : Forget gate, update gate, output gate

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

a_t 나 a_<t-1>, c<t-1>을 더 많이 사용

이전 time step 에서 전달받는 input 이 c<t-1>, a<t-1>로 추가됨

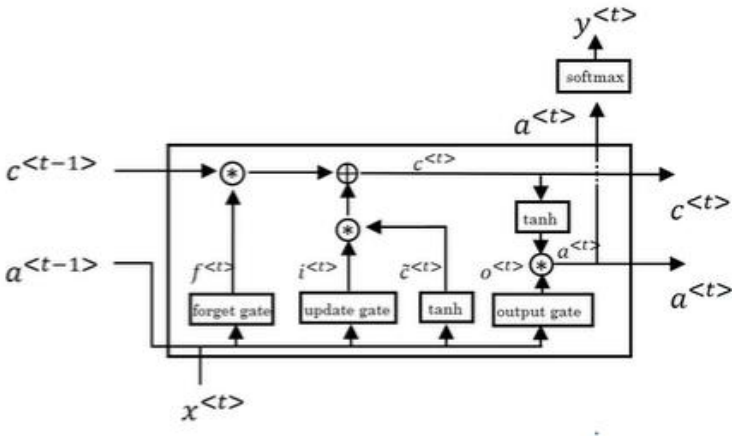
~c<t> : 현재 time step 에서 다음 time step 으로 업데이트할 정보들의 후보군

gamma_f = 1 – gamma_u

c<t> : 메모리 셀에 기존 값인 c<t-1>을 유지하는 옵션 만들고, 새 값 ~c<t> 더함

Forget gate : 현재 time step 의 정보를 바탕으로 이전 time step 정보를 얼마나 잊을지 정해 줌

그 결과에 현재 time step 의 정보의 일부(update gate 와 ~c<t>의 연산결과)를 더해서 다음 time step 으로 정보 전달

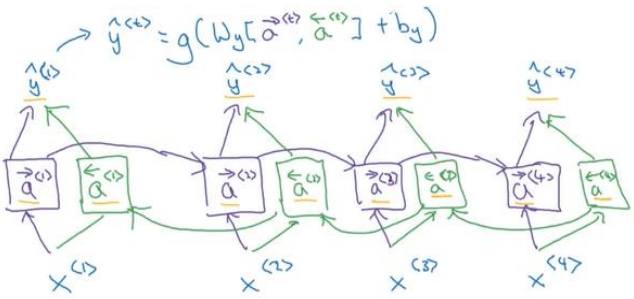


위의 일렬 선

: forget gate 와 update gate 과정 => 특정 값을 암기하는데 뛰어남

** peephole connection : 연산에서 a<t-1>, x<t> 대신 c<t-1>로 대체 ; 게이트 값이 이전 메모리 셀 값이 달려 있음

[Bidirectional RNN(양방향 RNN)] : 시퀀스에서 한 시점의 전과 후의 정보 모두 사용



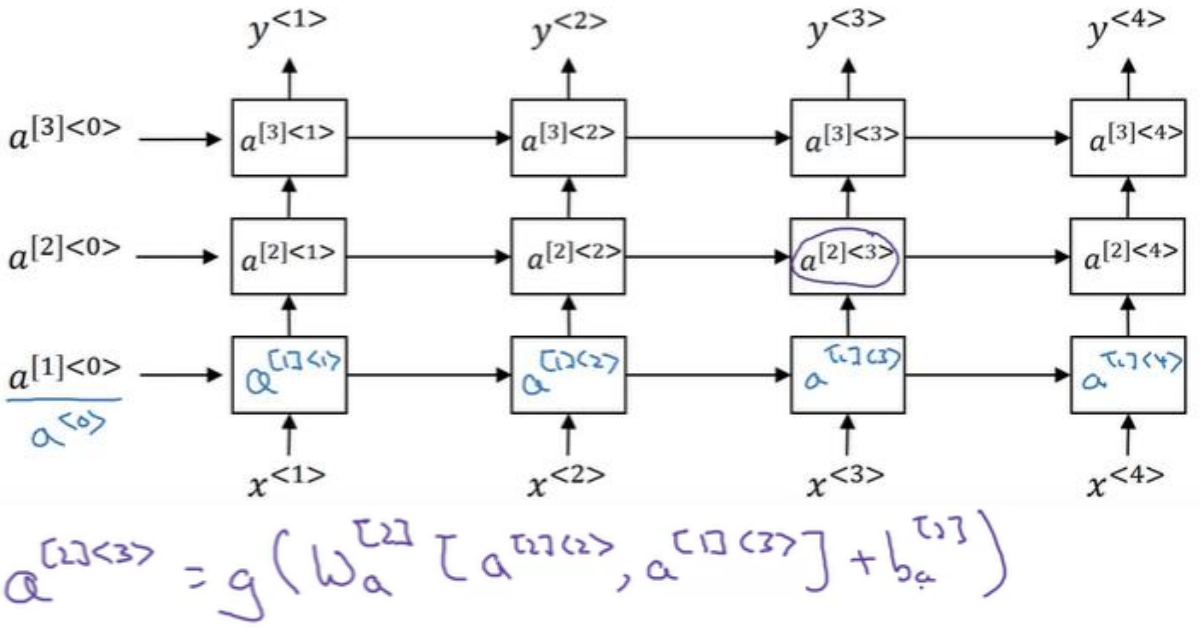
보라 cell : 앞에서부터 정보를 읽어 나감

초록 cell : 역방향으로 입력 정보 읽어 나감

=> 과거와 미래 정보 모두 참조하여 예측

단점 : 예측을 하기 전 전체 데이터 시퀀스 필요 (연설에서 음성을 다 듣고 예측)

[Deep RNN]



[Word representation]

1) 1 – hot representation

각 단어를 하나의 개체로 여기고, 알고리즘이 교차 단어를 쉽게 일반화하는 것을 허용하지 않는다는 단점 : 단어 간의 관계를 추론할 수 없음

ex) I want a glass of orange _____ / I want a glass of apple _____

빈칸에 juice 가 들어가도록 학습해도, apple 을 orange 와 비슷하다고 해 juice 를 추론할 수 없음

두 개의 서로 다른 one-hot vector 사이에서의 곱셈 결과는 0 이기에 관계를 학습할 수 없다.

2) Featurized representation : Word Embedding

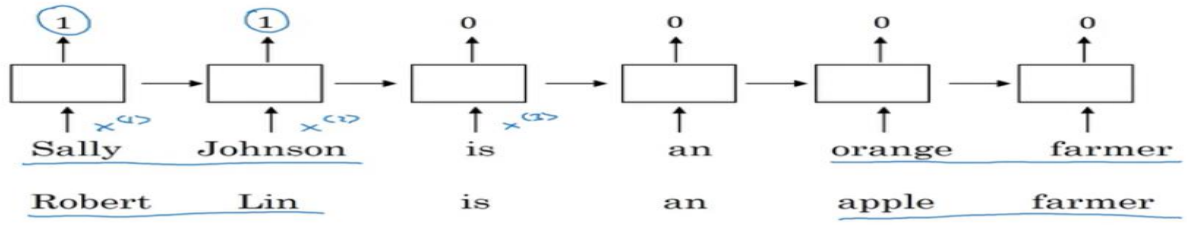
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

Man 같은 변수들은 현재 4 차원 벡터 : (-1, 0.01, 0.03, 0.04) -> e_5391

Apple 과 orange 가 비슷한 특징을 갖고 있을 것이라고 예측

⇨ 시각화 작업 : t-SNE 알고리즘 ; 임베딩 행렬을 더 낮은 차원으로 매핑해서 시각화

[Using Word Embeddings]



위의 문장에서 Sally Johnson 이 이름이라는 것을 확실히 하기 위한 방법 중 하나는 orange farmer 가 사람임을 알아내는 것 -> one-hot encoding 이 아닌 word embedding 을 사용해서 학습한 후, 새로운 example(밑의 문장)에서 apple 이 orange 와 유사하다는 것을 알기에 Robert Lin 이 사람 이름이라는 것을 더 쉽게 예측할 수 있다.

IF apple farmer 가 아닌 많이 사용되지 않는 과일인 durian cultivator 라면 ?
-> training set 이 적고, 이 안에 durian, cultivator 가 포함되지 않을 수 있다.
durian = 과일, cultivator = 사람 나타낸다는 것을 학습한다면, orange farmer 에서 학습한 것을 durian cultivator 에도 일반화하게 될 것이다.

Word Embedding 이 이렇게 일반화할 수 있는 이유 중 하나는 학습하는 알고리즘이 매우 큰 단어 뭉치들을 통해 학습하기 때문 + 적은 수의 training set 을 가지고 있어도, transfer learning 을 통해 미리 학습된 word embedding 을 가지고 학습할 수 있다.
1) large text corpus 로부터 word embeddings 배움 (pretrained embedding online 다운로드 가능)
2) transfer embedding to new task with smaller training set => 저차원 feature vector 사용
3) (Optional) Continue to finetune the word embeddings with new data (new data 학습할 때 더 작은 레이블 데이터셋을 사용하여 엔티티 인식 작업에서 필요한 것은 계속 finetuning 하고 조정)
-> 2)에서 데이터셋이 아주 큰 경우에만 수행 가능
=> NLP 작업에서의 word embedding
/ Transfer Learning 처럼 1)의 데이터는 많고, 2)의 데이터는 적을 때 효과적

이미지는 처음보는 이미지더라도 encoding 가능

단어 임베딩의 경우, 입력으로 사용되는 단어 정해져있어 UNK 과 같은 알 수 없는 단어(voca 에 존재하지 않는)가 포함 -> 정해진 단어만 학습함

[Properties of Word Embeddings]

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

e_5391 = e_man

Man -> Woman , King -> __?__

$$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

man 과 woman 의 관계가 king 과 queen 의 관계와 유사하다고 추론 가능

t-SNE 알고리즘 : 300D -> 2D 로 매핑 ; 임베딩을 통해 단어간의 관계 추론할 때, t-SNE 를 통해 매핑된 임베딩 값으로 비교하면 안되고, 300D 의 vector 를 통해 비교 연산 수행해야함

$$\textbf{Find word } w: \arg\max_w \text{sim}(e_w, e_{king} - e_{man} + e_{woman})$$

두 단어 사이의 유사성 ⬆

$$sim(e_w, e_{king} - e_{man} + e_{woman})$$

$$\|u - v\|^2$$

⬆ 유클리디안 거리

$$Sim(u,v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

⬅ 코사인 유사도

[Embedding Matrix]

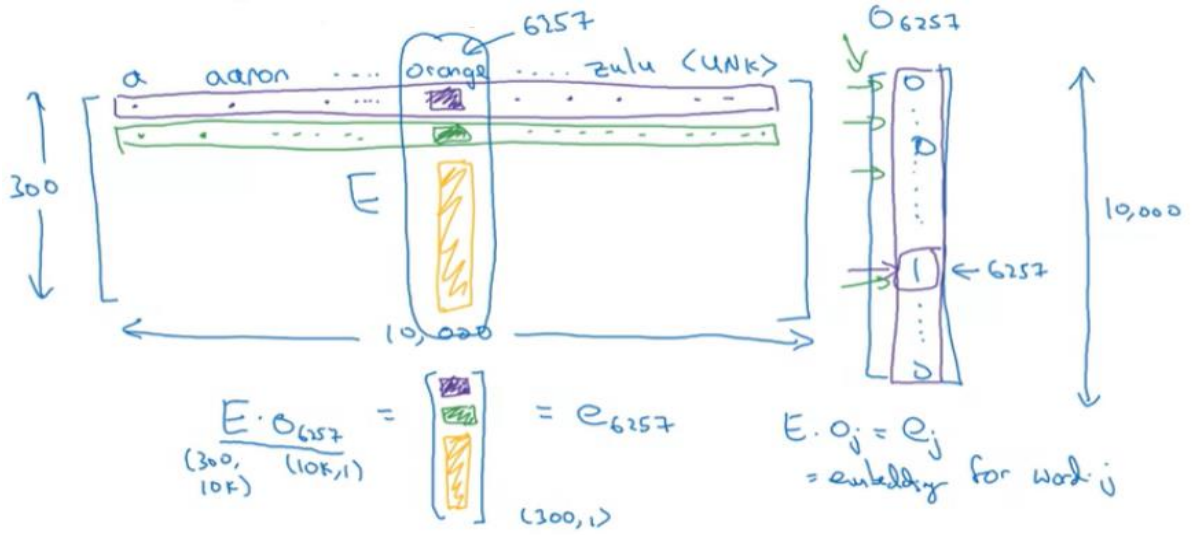
1 만개 단어 사용하고, 특징으로 300 차원 사용한다면, (300, 10000) 차원의 matrix E 를 가짐

voca 에 담겨있는 10000 개의 단어들을 각각 다르게 임베딩

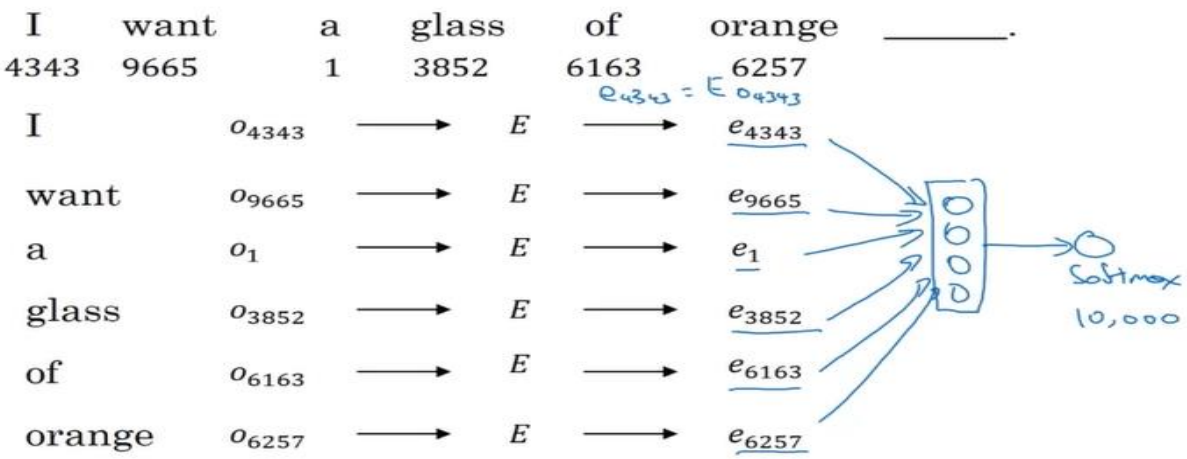
one-hot encoding 을 통한 하나의 단어에 대한 임베딩 vector 를 matrix E 와 내적(dot product)를 수행하면, 우리가 원하는 열(단어에 대한 임베딩)의 embedding vector 를 얻을 수 있다.

=> E * o_j = e_j

E : 초기에 무작위로 초기화됨 / 실제로는 one-hot-vector 를 곱하지 않음(메모리 낭비, 큰 연산량)



[Leaning Word Embeddings]



o_j : 10000 차원 벡터 / e_j : 300 차원 벡터

신경망과 softmax layer 는 각각의 weight 와 bias 가 있고, 각 단어들은 300 차원의 vector 이므로 입력은 총 1800 차원(300 * 6 개단어)의 벡터가 된다.

** Fixed historical window : 다음 단어를 예측하는데 그 단어 앞의 4 개(파라미터)의 단어만 사용

-> 이것은 입력으로 1200 차원(300 * 4 개 단어)의 vector 를 이용하여 softmax output 으로 예측

이 모델의 매개변수는 matrix E / 알고리즘의 매개변수는 w[1], b[1](신경망) , w[2], b[2](softmax)

[Word2Vec] : 단어 -> vector 로 바꿔주는 알고리즘

- Skip gram : 중심이 되는 단어를 무작위로 선택하고, 주변 단어를 예측하는 방법

I want a glass of orange juice to go along with my cereal.

Context(중심단어) c : orange / orange / orange

Target(예측 값) t : juice / glass / my (여러 개 선택 가능)

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

<- softmax output

세타_t : output 과 관련된 weight parameter (bias 생략)

$$\mathscr{L}(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i log\hat{y}_i$$

<- Loss (negative log likelihood)

y_hat : 10000 차원의 단어의 확률

y : one-hot-vector 10000 차원

Skip gram 의 softmax model 경우, 계산 속도 문제 존재 !

=> 해결 방법 : hierarchical softmax(계층적 softmax) ; tree 사용

자주 사용되는 단어일수록, tree 의 top 에, 그렇지 않다면 bottom 에 위치

; 선형 크기가 아닌 voca 사이즈의 log |voca| 사이즈로 탐색하게 되어 빠름

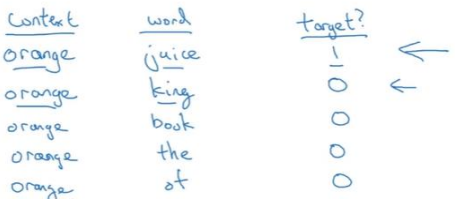
- Negative Sampling : c 를 샘플링하면, t 를 context 의 앞뒤 10 단어 내에서 샘플링 가능

c : 무작위로 균일하게 샘플링 -> the, of, a, to 등이 빈번하게 샘플링됨

=> 빈번한 단어들과 그렇지 않은 단어들 간의 균형 맞춰야 함

Skip gram 보다는 좀 더 빠른, 효율적인

I want a glass of orange juice to go along with my cereal.



Orange 와 juice 단어 한 쌍이 주어지면, 이것이 문맥 대상 쌍인지 예측할 수 있다. -> 긍정(1)

orange(context)와 king(word) -> 부정(0)

Orange juice 같은 positive training set 이 있다면, 무작위로 negative training set 을 K 개 샘플링

-> negative 단어를 선택할 때는 voca 에 존재하는 단어에서 무작위 선택 (여기선 K=4)

-> corpus 에서 empirical frequency(경험적 빈도)에 따라 샘플링 ; 문제는 the, a, of 같은 단어들 자주 등장

-> 극단적인 방법은 1/voca size 사용하여 무작위로 샘플링 ; 영어 단어의 분포 생각하지 않음

하지만, 우연히 'of' 단어를 선택할 수 있는데, 이는 context 에 존재하는 단어이므로 실제로는 positive 이지만, 일단 negative 로 취급

알고리즘이 단어 쌍을 x 개 입력하고 목표 레이블을 예측하여 출력 y 를 예측하는 지도학습 문제
작은 데이터셋의 경우 K 를 5~20 의 값으로, 큰 데이터셋의 경우 K 를 2~5 의 값으로 훈련셋 생성

Context, word(target)이 input 인 x 가 되고, positive/negative 는 output y => Logistic Regression => softmax 가 아니라 이진분류로 계산량이 훨씬 줄어들고,
한 개의 positive 와 K 개의 negative 샘플만 학습 (10000 개의 이진분류 문제 -> K+1 개만 학습)

[GloVe Word Vectors] : 모멘텀 가지는 word embedding 의 알고리즘

corpus 에서 context 와 target 단어들에 대해 i 의 context 에서 j 가 몇 번 나타내는지(X_ij) 구함
서로 가까운 단어를 캡처하며, context 와 target 의 범위를 어떻게 지정하느냐에 따라
X_ij = X_ji 가 될 수도(context 와 target 이 각각의 10 개 내외 단어 내에 나타나는지),
안될 수도(context 가 target 단어 바로 앞의 단어일 때) 있다.

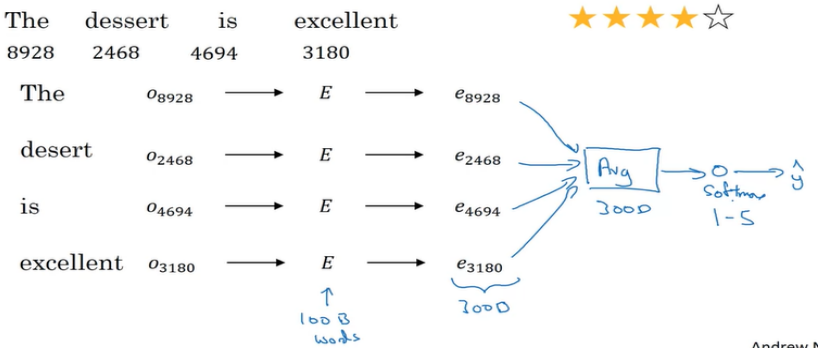
X_ij 는 c(i)와 t(j)가 얼마나 연관성이 있는지 포착하는 수

$$\text{minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij})(\Theta_i^T e_j + b_i + b'_j - log X_{ij})^2$$

이를 최적화해주는 것 / f(X_ij) : 가중치, 세타_i 와 e_j 는 대칭적

이 알고리즘을 훈련하는 방법 ; 최소화하기 위해 경사하강법에 맞춰 세타와 e 를 동일하게 초기화
이를 모든 단어에 대해서 수행하고 평균을 얻음
최종 e_w = (e_w + 세타_w) / 2

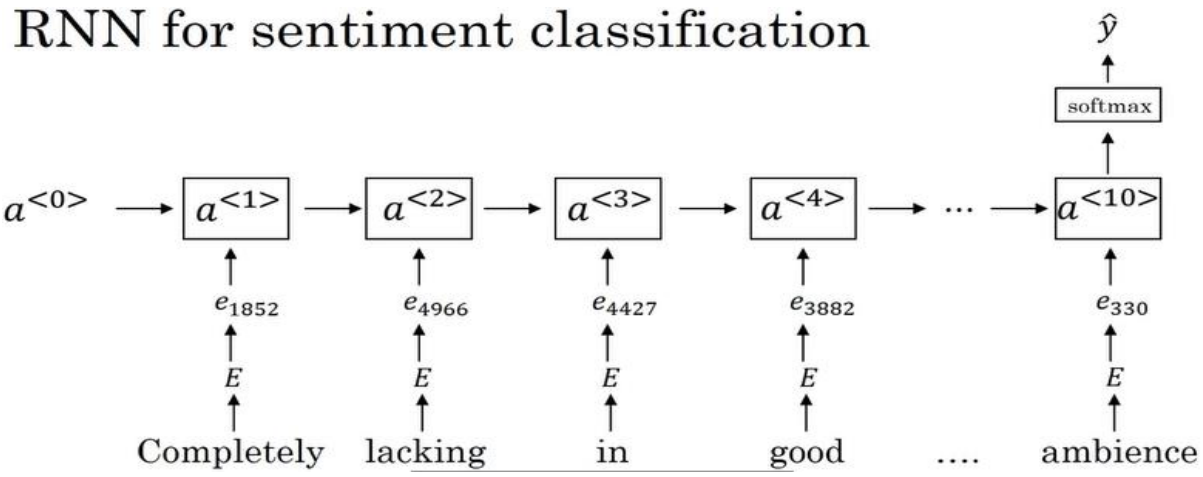
[Sentiment Classification] 감성분류 ; 큰 레이블 훈련셋이 없을 수 있지만, Word Embedding 사용하면 아주 작은 규모의 레이블 훈련셋만 있어도 좋은 감성 분류기 만들 수 있음



E : 1000 억개 단어 행렬
각 단어들을 300 차원인 임베딩 vector(e_)로 변환하고, 각 단어의 벡터의 값들의 평균을 구해 softmax output 으로 결과 예측

단어 임베딩을 사용했기에 작은 dataset 이나 자주 사용되지 않는 단어, 학습에 사용되지 않는 단어가 입력으로 들어와도 적용이 가능하다.

하지만, 단점으로 단어 순서를 무시한다는 것이다.
'Completely lacking in good taste, good service, and good ambience' 의 리뷰에서
good 이라는 단어가 많이 들어가 positive 로 예측할 수 있지만, 리뷰의 내용은 좋은 맛, 좋은 서비스, 좋은 분이기가 없다는 부정적인 리뷰이다.
=> RNN 사용한 감성 분류 (many – to – one)

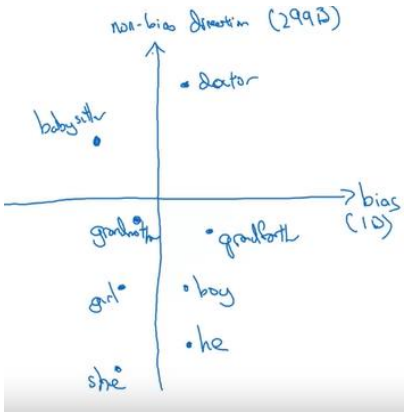


[Debiasing Word Embeddings]

여기서 말하는 bias 는 성별, 민족, 성적, 성향, 사회 경제적 지위와 같은 편향(편견)을 뜻함

Word Embedding 은 모델을 훈련하는 데 사용되는 텍스트의 편견들을 반영할 수 있다. 이런 편견을 제거하기 위해, 다음과 같은 과정을 거쳐 제거할 수 있다.

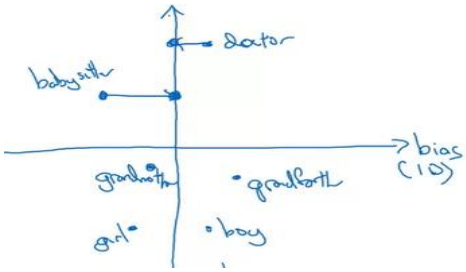
1) 특정한 편견에 해당하는 direction(방향)을 파악한다.



(e_he – e_she)와 (e_male – e_female) 같은 차(-)를 가지고 평균을 내고, 이 방향으로 보이는 것이 성별로 편향된 방향이다. 편향 방향은 1 차원 부분 공간이 되고, 비편향 방향은 299 차원 부분 공간이 된다.

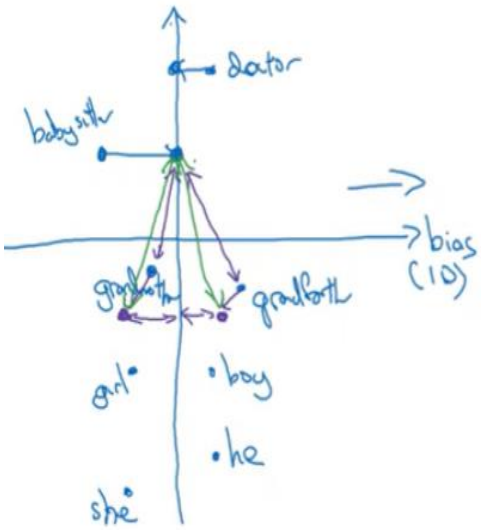
편향 방향은 1 차원보다 고차원이 될 수 있고, 평균을 취하는 대신 실제로는 SVD(특이값 분해)를 통해 발견된다.

2) 중립화 단계(Neutralize) : 명확하지 않은 모든 단어의 경우 추정하여 bias 제거



Bias 축 밑으로의 단어들은 정의에 성별이 내재되어 있다. 하지만, 그 위의 단어(베이비시터, 의사)는 중립적이길 원한다.
각 단어 벡터의 bias direction 요소를 제거한다. Y 축 방향으로 수평적으로 거리 이동

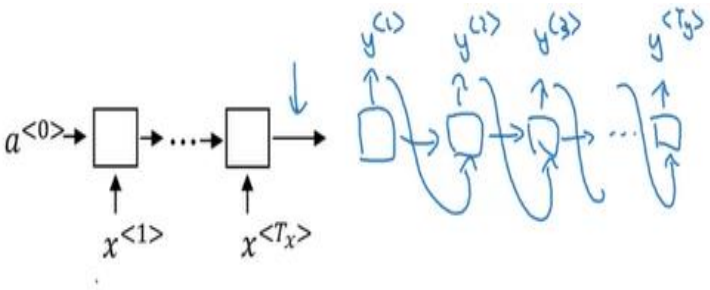
3) Equalize pairs



boy – girl / grandfather – grandmother 과 같은 단어는 각 단어가 성별 요소가 있기에, bias direction 을 기준으로 같은 거리에 있도록 한다.
즉, 각 성별 요소가 있는 단어들은 bias direction 과의 거리의 차이가 동일하도록 만들어 준다.

Doctor 는 grandmother/girl/she 까지의 거리보다 grandfather/boy/he 까지의 거리가 더 가깝다. 이는 편견을 강화할지 모른다. 그래서 마지막 평준화 단계에서 중립적인 단어로부터 정확히 같은 거리에 있거나 유사성이 같은 지 확인

[Basic Models] ; 시퀀스 대 시퀀스 모델 – 무작위로 선택된 것을 원치 않고 정확한 것을 원함

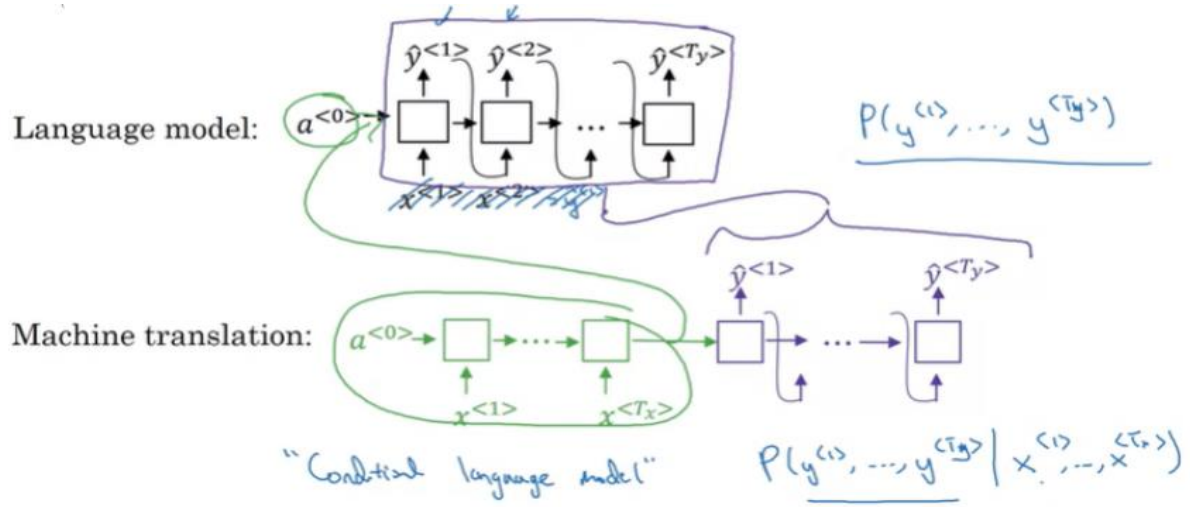


인코더 네트워크를 통해 프랑스어 단어를 입력 받고, 디코더 네트워크를 통해 영어 단어 하나하나를 출력

이 모델은 충분한 프랑스어, 영어 문장의 dataset 이 있을 때 효과적

➔ Image captioning 에도 사용 (input : 이미지, output : 자막)
pretrained 된 AlexNet 사용해서, 마지막 softmax 제거하면 이미지를 나타낼 수 있는 4096 차원(AlexNet) 특성벡터 제공 (인코더 네트워크)
-> 한번에 한 단어씩 자막을 생성하는 RNN 으로 전달 (RNN 을 디코더 네트워크로 사용해 특성벡터를 입력으로 사용)

[Picking the most likely sentence]



Machine translation 은 뒤쪽의 디코더 네트워크가 language model 과 유사하다.
language model 은 0 벡터에서 시작했지만, Machine translation 모델에서는 인코더 네트워크를 통과한 출력이 language model 로 입력되는 것과 같다.
=> Conditional language model (조건부 언어모델)

주어진 x 에 대한 y 의 확률이 높은 것을 원하는데, 무작위로 샘플링하지 않고, 조건부 확률을 극대화하는 y 를 찾기를 원한다. 항(y<1>, ...)을 최대화하는 y 값을 찾을 수 있는 빔 알고리즘

** 탐색적 알고리즘(greedy search) : 조건부 언어 모델에 따라 가장 가능성이 높은 첫 번째 단어를 선택하여 첫 번째 단어를 생성하고, machine translation 으로 이동한 다음 첫 번째 단어 선택한 후, 가장 가능성이 있는 두 번째 단어를 고르고 가장 확률이 높은 세 번째 단어 선택
; 전체 단어 시퀀스를 선택함, 전체 단어의 동시 확률 극대화 => 이 알고리즘은 사용 X !!

마지막 단어까지 도달하며 가장 높은 확률의 문장을 찾아야 하는데, voca size 를 고려하면 모든 단어들을 평가하기에 불가능하다. => heuristics 탐색 알고리즘 활용하여 "대략적인" 최대치 찾음

[Beam Search] : 가장 확률이 높은 output 을 찾기 위해 사용되는 알고리즘

B (beam width) = 3 : 3 개의 가장 높은 가능성을 고려 / B = 1 이면, greedy 알고리즘

Step 1) 가장 확률 P(y<1> | x)이 높은 단어 3 개를 선택
: 인코더 네트워크를 통해 프랑스어 문장을 입력한 다음, 디코더 네트워크의 소프트맥스 출력을 얻고, 이 중 상위 3 개를 메모리에 보관

Step 2) P(y<1>, y<2> | x) = P(y<1> | x) * P(y<2> | x, y<1>)

[Refinements to Beam Search]

- Length normalization

$$\arg\max_{y} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

P 는 확률이기에 1 보다 작아 P 를 곱하게 되면 그 값은 훨씬 작아지게 된다. 그러면 모델이 짧은 번역을 더 선호하게 되는 문제 발생할 수 있다. -> log 취하기 !
하지만 확률은 1 보다 항상 작거나 같기에, 더 많은 단어가 있을수록 음수가 점점 커지게 된다.
=> 단어의 수 T_y 로 normalization 하기 !

$$\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

T_y 가 큰 경우의 페널티를 감소하기 위해 alpha 승을 사용할 수 있다. ; normalized log probability

- Beam width, B

large B : 여러 선택지 시도하기에 더 나은 결과 / 속도는 느리고, 메모리 요구량 증가, 느린 계산
small B : 빠른 속도, 메모리 요구량 적음 / 적은 가능성만 고려하기에 좋지 않은 결과

[Error Analysis in Beam Search]

Beam Search 는 추론적 검색 알고리즘이라고도 불리는 대략적인 검색 알고리즘(휴리스틱)

Jane visite l'Afrique en septembre.

Human: Jane visits Africa in September. (y^*)

Algorithm: Jane visited Africa last September. ($\hat{y}$)

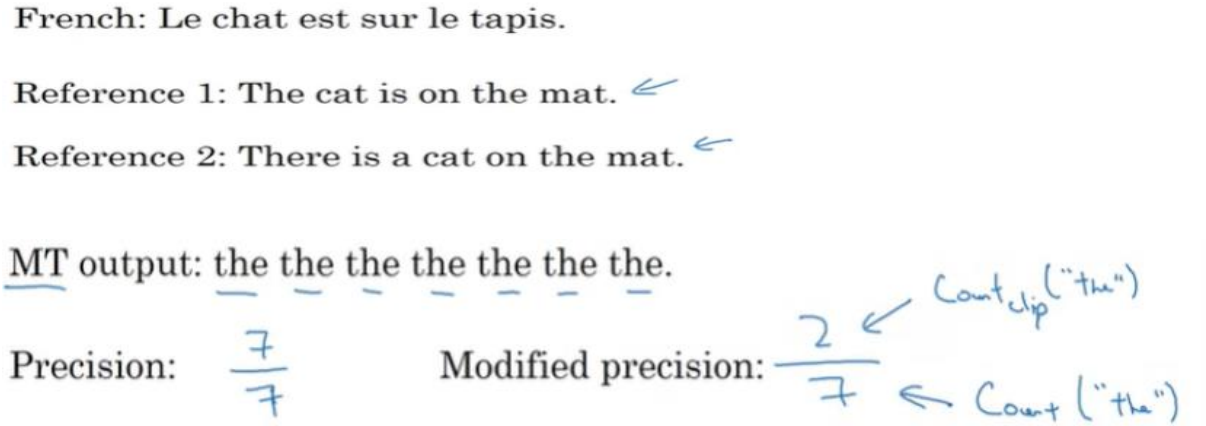
P(y* | x)와 P(y 햇 | x) 계산하여 큰 값 확인하는 것으로 모델의 오류 (RNN 의 문제인지, Beam Search 의 문제인지)를 명확하게 설명할 수 있다.
P(y* | x) > P(y 햇 | x) : Beam Search 가 더 높은 확률의 번역을 선택하지 못함
P(y* | x) <= P(y 햇 | x) : RNN 모델이 잘못된 번역을 더 높은 확률로 예측

Beam Search 가 많은 오류를 유발한다면, Beam width(B)를 늘림
RNN 모델에 문제가 있는 경우, 더 깊이 있는 분석 계층을 통해 정규화를 추가하거나 더 많은 훈련 데이터를 얻거나 다른 네트워크 아키텍처 등을 시도해 볼 수 있음

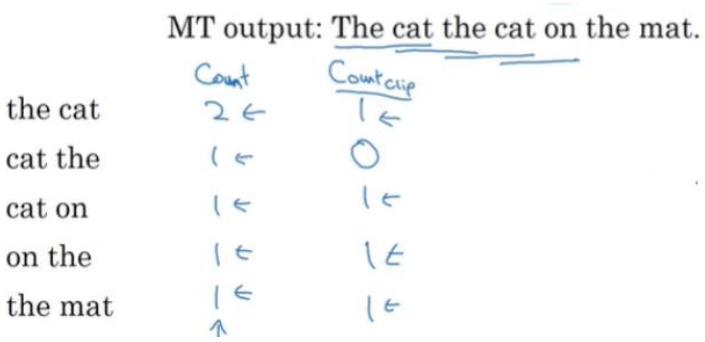
[Bleu Score]

하나의 정답이 있는 이미지 인식과는 달리, 여러가지의 좋은 정답이 있다면 Machine translation 은 정확성을 측정하여 평가하는데, 보편적으로 Bleu Score 방법을 사용한다.

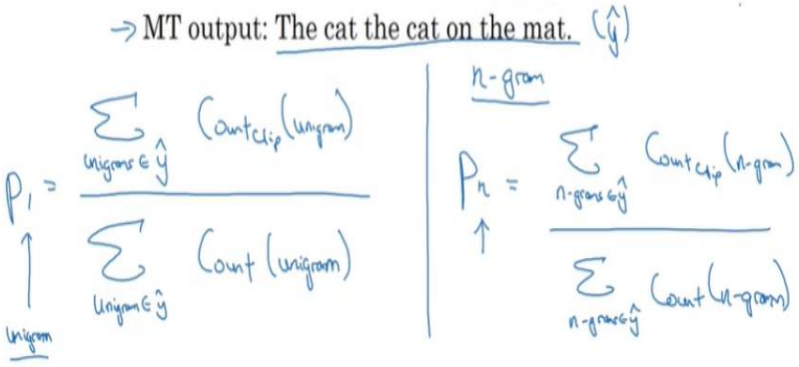
모델이 예측한 결과가 사람이 제공한 reference 와 가깝다면 Bleu score 는 높게 된다. Bleu Score 는 직관적으로 기계가 생성하는 글의 유형을 최소한 인간이 만들어낸 reference 에서 나타나는지 살펴보는 방법이다.



Precision : 각 단어를 보고 reference 안에 그 단어가 존재하는지
Modified Precision : 각 단어에 대해 중복을 제거하고, reference 문장의 최대 횟수만큼 점수 부여 (ref1 에서 the 는 2 번 등장, ref2 에서 the 는 1 번 등장 -> 2)
=> 각 단어를 개별적(isolated word)으로 살펴보고, 단어의 순서를 고려하지 않았다.



biagrams 에서는 MT output 의 각 단어들을 서로 근접한 두 개의 단어들로 묶어서, reference 에 얼마나 나타나는지 체크하여 Modified Precision 계산
= 4 / 6



<- unigrams, n-grams
MT output 이 ref1 이나 ref2 와 정확히 같다면, P1, P2 는 1.0 과 같다.

$$BP = \begin{cases} 1 & \text{if MT\_output\_length} > \text{reference\_output\_length} \\ \exp(1 - \text{reference\_output\_length / MT\_output\_length}) & \text{otherwise} \end{cases}$$

최종 Bleu Score 만들기 위해 P_n(n-grams 에서 bleu score)을 합쳤다.

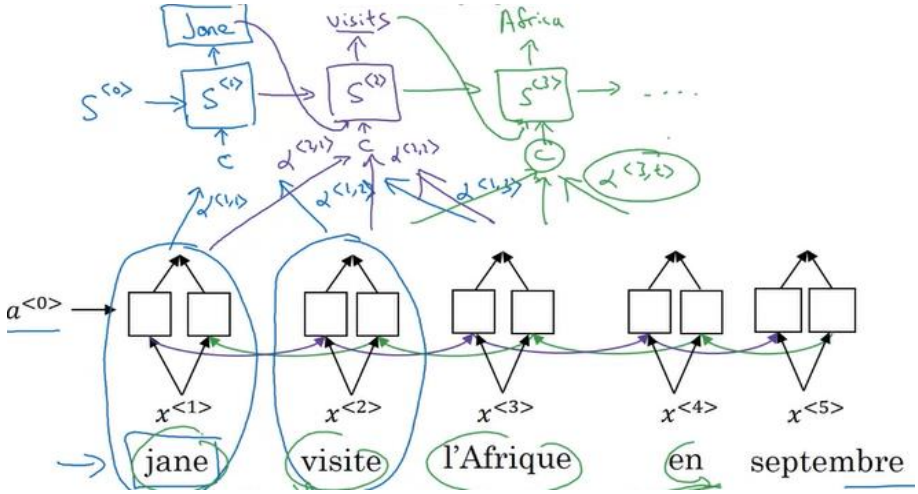
BP(Brevity Penalty) : 짧은 번역이 높은 score 을 얻는 것에 대해 페널티 부여

$$\text{Combined Bleu score: } BP\exp\left(\frac{1}{4}\sum_{n=1}^{4} p_n\right)$$

1, 2, 3, 4-grams 의 최종 bleu score

[Attention Model]

- 인간 번역자 : 첫 부분을 읽고 번역의 일부를 만들어 내고, 두 번째 부분을 보고 몇 단어를 생성하고, 몇 단어를 더 살펴보고 생성한다. -> 문장 전체를 부분적으로 작업
- Encoder – Decoder : 하나의 RNN 에서 입력 문장을 읽고, 다른 RNN 에서 문장을 출력 이는 짧은 문장에서 잘 동작하고, 입력 문장 길이 길수록 성능이 낮아져 Bleu score 이 낮아진다.
- Attention : 긴 문장에서도 성능을 유지할 수 있다. 사람과 유사하게 문장을 외워 처음부터 번역하는 것이 아닌 중간중간 번역한다. 매 예측시점마다 인코더에서의 입력 문장을 다시 참고하는데, 이때, 전체 입력 문장을 참고하는 것이 아닌 예측할 단어와 연관되는 부분만 집중해서(Attention) 참조한다.



s : 위쪽 RNN 의 은닉 상태 activation (a 대신 s 로 표기)
alpha :어텐션 가중치

c : alpha 들이 모여 주의해야 할 정확한 맥락 알려주고, RNN 유닛에 입력되어 첫 번째 단어 생성 시도

alpha(t, t') : 영어 단어 t 를 만들 때, 얼마나 많은 t' 프랑스어 단어를 사용해야 하는지 가중치
(alpha<1, 1> : 첫 번째 단어 생성시, 첫 번째 정보에 얼마나 주의를 기울여야 하는지,
alpha<1, 2> : 첫 번째 단어 생성시, 두 번째 정보(단어)에 얼마나 주의를 기울여야 하는지)

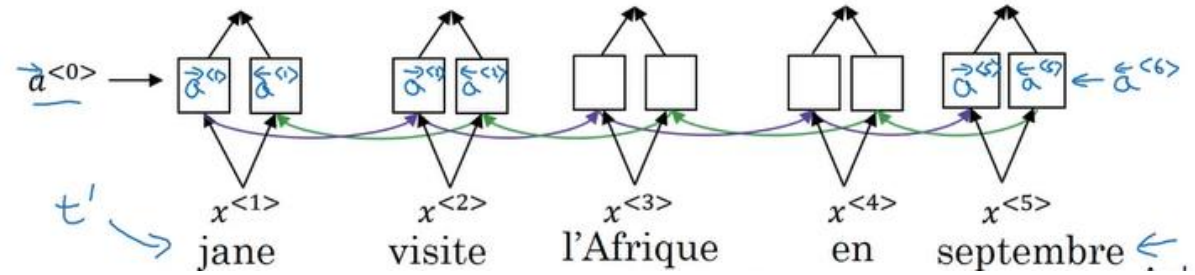
a 로 표기되는 아래 그림은 Encoder 에 해당하는 RNN model(BRNN),

s 로 표기되는 위 그림은 Decoder 에 해당하는 RNN model

Encoder(BRNN)을 통해 계산된 activation 을 사용해 c 를 구하는데, 이 때 일부 activation 만 참조

c 를 Decoder 의 입력으로 사용하여 단어 예측

EOS 일 때까지 한번에 한 단어를 만드는 RNN 의 순방향, 매 단계마다 어텐션 가중치가 있다.



: Encoder Network (BRNN)

순방향/역방향에서 계산되는 activation 은 합쳐 $a^{<t'>} = (\overrightarrow{a}^{<t'>}, \overleftarrow{a}^{<t'>})$ 로 표기

(t' : x 의 time step)

$$c^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$$

Decoder 의 입력 c

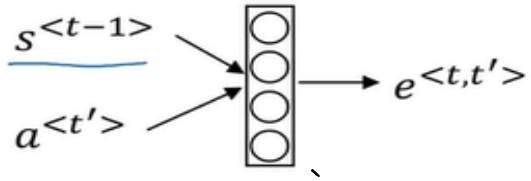
alpha : context 가 activation 과 feature 에 얼마나 의존적인지

y<t> 예측에 얼마나 attention 해야할 지 나타내는 정도

$$\alpha^{<t,t'>} = \text{amount of attention } y^{<t>} \text{ should pay to } a^{<t'>}$$

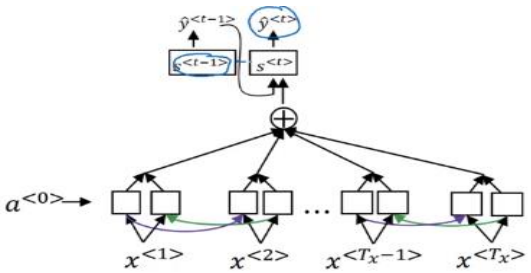
$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

alpha 는 소프트맥스 확률로 계산되기에 총합이 1 이 된다.



e<t, t'>는 Dense layer 을 통해 구할 수 있으며,

이전 layer 의 hidden activation s<t-1>, a<t'>를 입력으로 함 (s<t-1>, a<t'>가 Dense layer 에 들어가 e<t, t'> = alpha<t, t'>를 output 으로 가짐)



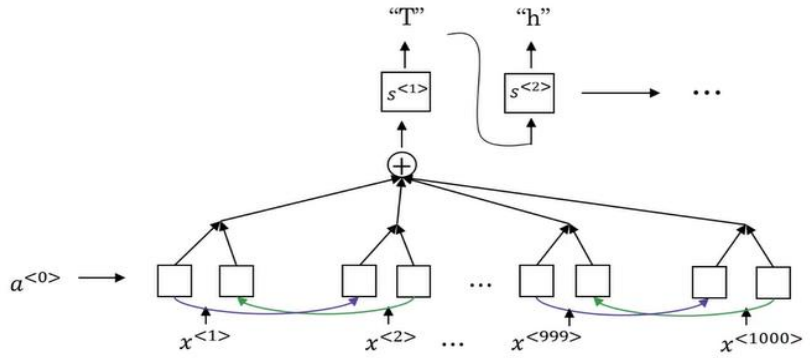
y_t 가 t'의 활성화 (alpha<t'>)에 얼마나 주의를 기울일 지를 결정한다면, 가장 많이 의존해야 할 것은 이전 layer 의 은닉 상태의 활성화 (s<t-1>)이다.

Attention model 은 어텐션 가중치 합이 1 이 되도록 한 다음, 느리지만 꾸준하게 한번에 한 단어를 생성하도록 한다. 이 신경망은 경사하강법을 이용하여 모든 것을 자동으로 학습하는 입력 문장의 오른쪽 부분에 주의를 기울인다.

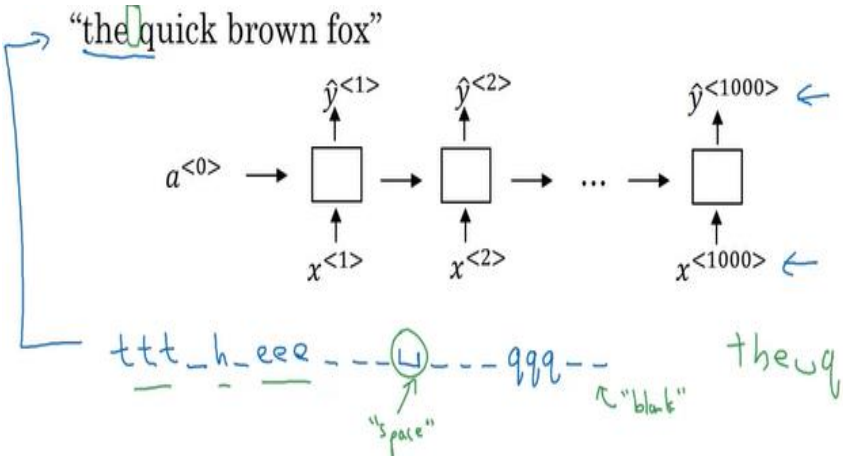
Attention model 의 단점) 학습시간(연산량)이 quadratic time(cost)을 가진다.
입력에 단어 T_x 와 출력에 단어 T_y 가 있다면, attention 매개변수 총 개수는 T_x * T_y 개이다.

[Speech Recognition]

x (audio clip) → y (transcript) : 입력 데이터를 주파수별로 분리



Attention model 을 음성 인식 모델에 사용할 수 있다.
: 가로축에서 다양한 시간 단계의 오디오를 입력한 다음, 음성을 글로 옮겨 출력하려고 시도



CTC 비용 사용 (Connectionist temporal classification)
: 양방향 LSP 와 양방향 GIU 모델에 사용, 음성 인식 시 일반적으로 입력 시간 단계의 수가 출력 시간 단계의 수보다 훨씬 크다.

10 초 분량의 오디오가 있고 특징이 100Hz 인 초당 100 개의 샘플이라면, 10 초의 오디오 클립은 천 개의 입력이 된다. (100Hz * 10 초) 하지만 출력 결과에 알파벳 천 개나 글자 천 개가 없을 수도 있다.
-> CTC 비용 함수를 사용하면 RNN 에서 'ttt_h_eee' 와 같은 출력을 얻을 수 있고, '_'는 blank 를 의미한다. CTC cost 의 기본 규칙은 'blank'로 구분되지 않는 반복된 문자를 축소하는 것(blank 는 글자를 나타내며 띄어쓰기와는 다름)이고, 예시에서는 output 이 'the q'가 될 것이다. 글자 반복을 통해 1000 개를 출력할 수 있다. 공백 문자를 많이 삽입하면 출력 텍스트 내용이 더 짧아진다.

[Trigger Word Detection] : 기계를 깨울 수 있는 방법

앞의 음성 인식 시스템은 아주 큰 데이터셋이 필요하다. 키워드 감지 시스템인 Trigger 단어 감지 시스템은 훨씬 더 적거나 합리적인 양의 데이터로 더 쉽게 작업이 가능하다.

- Trigger Word : 헤이 시리, 오케이 구글, 알렉사

Trigger word 를 말하기 전의 시점은 label 을 0 으로 설정하고, 말하고 난 직후 시점을 1 로 label
단점) 훈련셋 라벨은 1 보다 훨씬 더 많은 0 을 가짐(데이터 불균형)
=> 좀 더 쉽게 훈련시킬 수 있다. 단 한번이 아닌 label 1 의 비율을 늘려 출력하도록 할 수 있다.

[Transformer Network Intuition]

- 순차적 모델 : input 문장을 한번에 한 단어나 토큰을 소화, 마지막 단위의 output 을 계산하기 위해 이전에 오는 모든 단위의 output 을 계산하기에 각 단위가 정보 플로우에 걸림돌이 됨

RNN -> 기울기 소멸 문제로, 장거리 종속성과 시퀀스 캡처하는 데 어려움 발견
=> GRU, LSTM 모델을 정보 플로우를 제어하기 위해 gate 를 사용하여 문제 해결 -> 복잡도 증가

- Transformer : 전체 시퀀스에 대해 더 많은 계산을 병렬로 실행, 왼쪽에서 오른쪽으로 한번에 한 단어씩 처리하는 대신 전체 문장을 동시에 소화

Attention + CNN
- Self-Attention : 5 개의 단어로 된 문장이 있다면, 이 5 개의 단어에 대한 5 개의 표현을 계산하고, 문장의 모든 단어들에 대한 표현을 병렬로 계산하는 방법을 기반으로 하는 Attention
- Multi-Head Attention

[Self - Attention]

RNN Attention

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{Tx} \exp(e^{<t,t'>})}$$

Transformers Attention

$$A(q,K,V) = \sum_i \frac{\exp(q \cdot k^{<i>})}{\sum_j \exp(q \cdot k^{<j>})} v^{<i>}$$

RNN Attention 방정식과 유사, 분모는 softmax 로 표현

A(q, K, V) = attention-based vector representation of a word ; calculate for each word

q<t> : Query, 해당 단어인 X<t>에 대한 질문

k<t> : Key, 해당 쿼리에 대한 답에 대응하는 값

v<t> : Value, 최종 결과에 가중치 부여하는 값

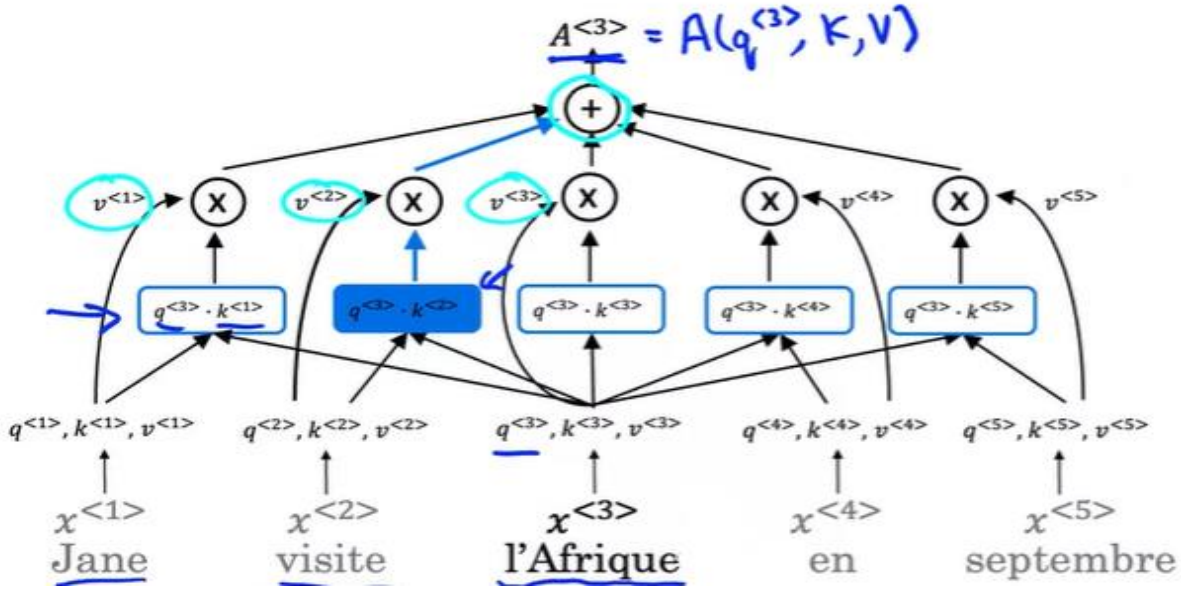
q<t>와 k<t>의 내적의 값이 클수록 큰 연관/맥락을 가진다. 모든 단어들에 대해 q(쿼리)와 k(키)를 내적한 결과에 v<t'>를 곱한 뒤, 더하여 최종 어텐션 A<t>를 계산하게 된다.

** q = W_Q * X

k = W_K * X

v = W_V * X

Ex) Jane visite l'Afrique on septembre(프랑스어)를 영어로 번역



k<1> : person k<2> : action

q<3>와 k<2>의 내적은 가장 큰 값을 가진 것을 볼 수 있고, 이는 Africa 에 무슨 일이 일어나고 있는지에 대한 질문(q<3> : What's happening?)에 visite 이 가장 적절한 맥락을 제공한다고 할 수 있다.(직관적) 즉, 방문지로 보고 있다.

Self-Attention 의 최대 장점은, 각 단어들이 고정된 word embeddings 가 아니다. Africa 는 visits 의 목적어로, 방문지를 의미한다는 것을 알게 된다. 즉, 더 풍부한 표현이 가능하다.

$$\boldsymbol{Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V}$$

Softmax 의 결과를 가중치로 이용하여, Value 를 이에 곱해 단어의 최종 가중치를 결정하게 된다.

** 단어 주어지면 해당 단어의 이웃 단어는 단어 값을 합산하여 해당 단어와 관련된 attention 을 매핑하여 해당 단어의 컨텍스트 계산

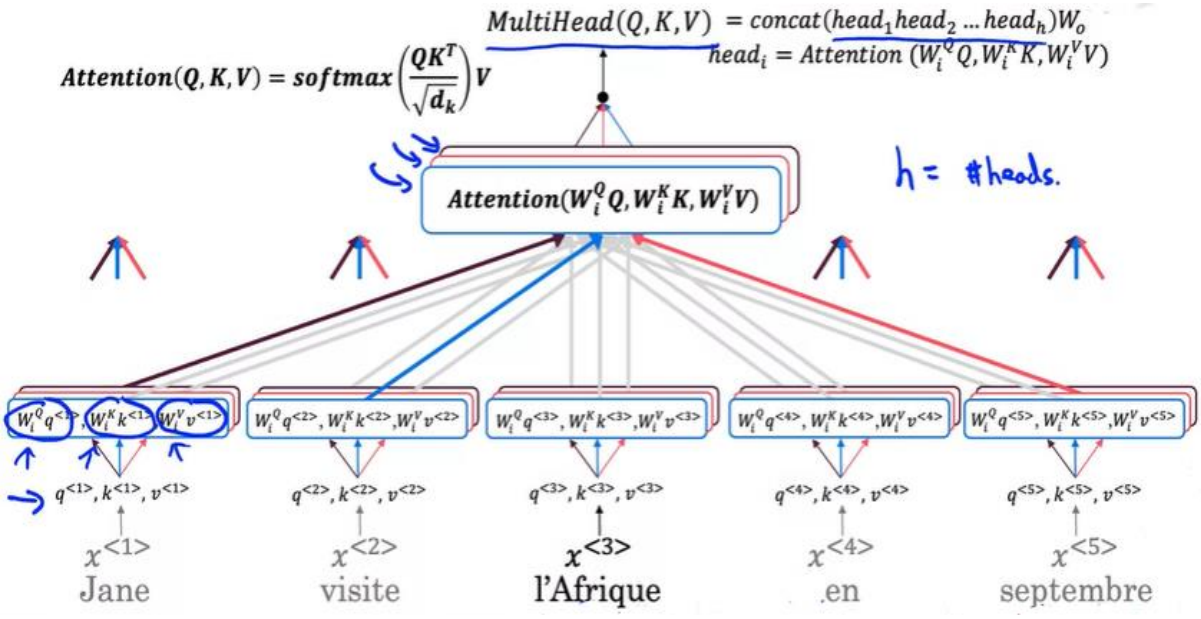
[Multi-Head Attention]

독립적인 여러 매커니즘에 대해 병렬하게 attention 을 수행
(Self-Attention 은 하나의 매커니즘에 대응)

동일한 쿼리(q), 키(k), 값(v) 벡터 세트를 입력으로 사용

각 단어는 고유한 q, k, v 를 가지고, 각 매커니즘에 대해 고유한 가중치 W1_Q, W1_K, W1_V 를 가지고, 각 단어의 q, k, v 에 곱한다. (W1_Q * q<1> / W1_K * k<1> / W1_V * v<1>)
각 Head 에는 이미 행렬 W 가 있으므로 여기서도 곱하기를 하면 계산을 두 번 할 수 있다. 여기서는 q = k = v = x (입력과 출력 사이의 attention 을 계산하는 곳에서는 학습할 때마다 다른 정보를 전달하기에 다른 값)
** Self-Attention 에서는 q, k, v 는 x 와 W 를 곱함

다른 매커니즘에 대해서는 또 고유한 가중치 W2_Q, W2_K, W2_V 를 가지고, W1 을 곱했던 attention 은 독립적으로 연산을 수행한다. 각 매커니즘에 대한 계산을 수행하는 것을 하나의 Head, 여러 매커니즘에 대해 병렬하게 수행하므로 Multi-Head.
(각 Head 는 병렬하게 연산 가능 ; 어느 것도 다른 헤드의 값에 의존하지 않음)



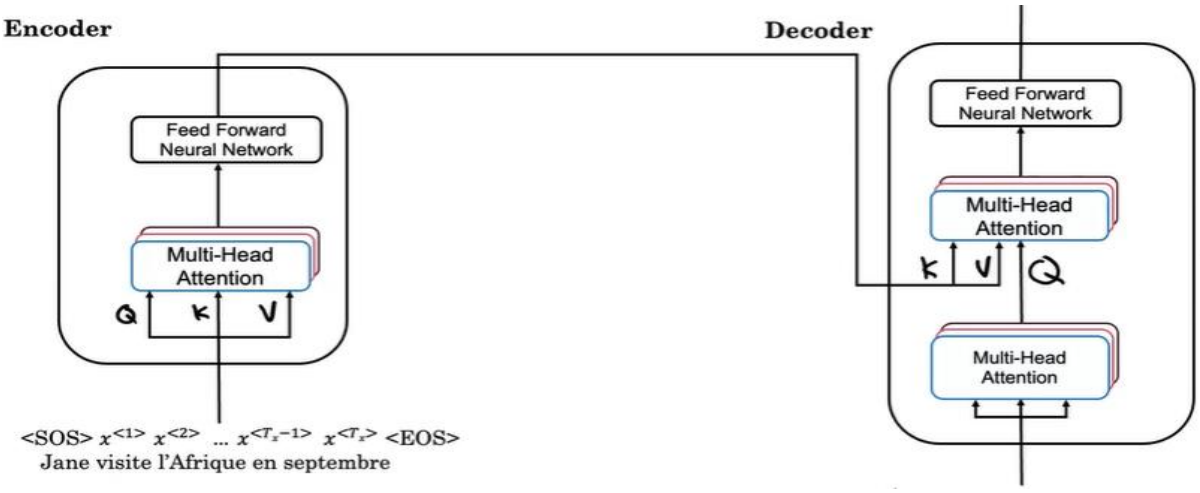
W1 : What's happening? -> l'Afrique 의 쿼리에 대해 가장 높은 값을 갖는 visite 의 키 (파란)
W2 : When ? -> l'Afrique 의 쿼리에 대해 가장 높은 값을 갖는 September 키 (빨간)
W3 : Who ? -> l'Afrique 의 쿼리에 대해 가장 높은 값을 갖는 Jane 키 (검은)

[Transformer Network]

Transformer : 인코더에서 입력 시퀀스를 입력받은 뒤, 디코더에서 출력 시퀀스를 출력하는 구조

- Encoder : 각 단어들이 전체 문장에서 어떤 특징을 가지고 있는지 계산하기 위한 Multi-Head Attention 과 Feed Forward NN 을 가짐, 이를 N 회 반복 (n 은 보통 6)
- Decoder : 시작 단어인 <SOS>로부터 출력 시퀀스의 다음 단어 번역을 예측하고, 그 값을 다시 디코더의 입력으로 넣어 그 다음 번역을 예측한다.

디코더의 입력으로부터 Q 값을 계산하고, 인코더로부터 K, V 값을 얻어 계산 수행



1) 임베딩을 Multi-Head Attention layer 가 있는 Encoder 블록에 공급

2) 임베딩과 가중치 행렬 W 에서 계산된 값 Q, K, V 를 입력한 레이어는 Feed Forward 신경망에 전달할 수 있는 행렬을 생성 : 문장에 어떤 특징이 있는지 판단

3) 2)를 n 번(약 6)반복하고, 인코더의 출력을 디코더 블록에 공급 ; 영어 번역 출력
   - 첫 번째 출력은 문장 토큰의 시작 부분
   - 시작 토큰 <SOS>가 입력되고 Q, K, V 를 계싼하는 데에는 <SOS>토큰 하나만 사용
   - 이 첫 번째 블록의 출력은 다음 Multi-Head Attention 블록의 행렬 Q 를 생성
   - 인코더의 출력은 K 와 V 를 생성

   (<SOS>는 지금까지 번역한 내용, 문장의 프랑스어 버전에서 번역된 K 와 V 에서 컨텍스트를 가져와 시퀀스에서 다음으로 생성할 단어 결정)

4) 이를 Feed Forward 신경망에 입력하여 출력함 -> n(약 6)번 반복

[Positional Encoding]

Transformer 의 Encoder 에서 중요한 것은, 입력 시퀀스가 한번에 들어오기에 각 단어 별 위치 정보를 알려줄 필요가 있다.

Self-Attention 은 단어의 위치를 나타내는 것은 없다. 하지만 문장 내 위치는 번역에 매우 중요

Positional Encoding

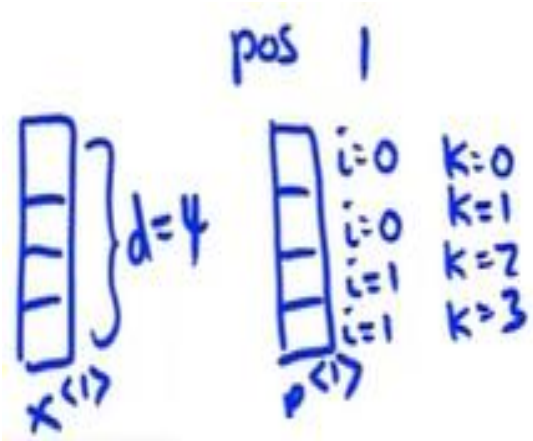
$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d}}})$$

입력 값의 요소 위치를 인코딩하는 방법은 sin 방정식과 cos 방정식을 조합하여 임베딩하여 위치 정보 알 수 있다.

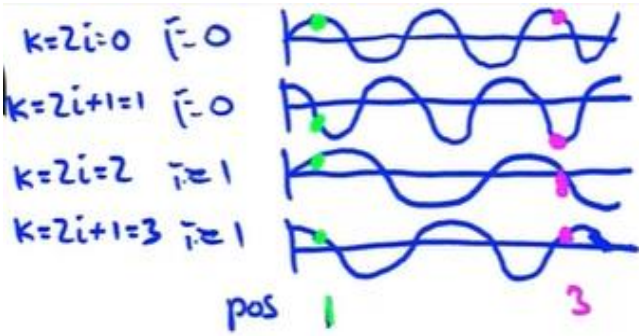
pos : 단어의 수치적 위치 / i : 인코딩의 다양한 차원

단어 임베딩 값이 4 개인 벡터 (단어 임베딩 차원 D = 4 / x<1> : 4 차원 벡터)
같은 차원인 4 차원의 위치 임베딩 벡터 p<1> ← 첫 번째 단어의 위치 임베딩을 Jane 으로 가정
Jane 이라는 단어의 경우 pos = 1



i=0, 0, 1, 1
: 두 번씩 반복하는 이유는 i 의 각 값을 사인과 코사인 함수를 사용하여 2 차원을 부호화하는데 사용하기 때문
i 를 계산하기 위해 도우미 인덱스 k 를 사용할 수 있는데, 이는 단순히 단어 임베딩 차원을 0 에서 d-1 까지 센 다음 i 를 k/2 의 정수 나눗셈으로 계산
k = 2i + 1 = 1 / k = 2i = 2 / k = 2i+1 = 3

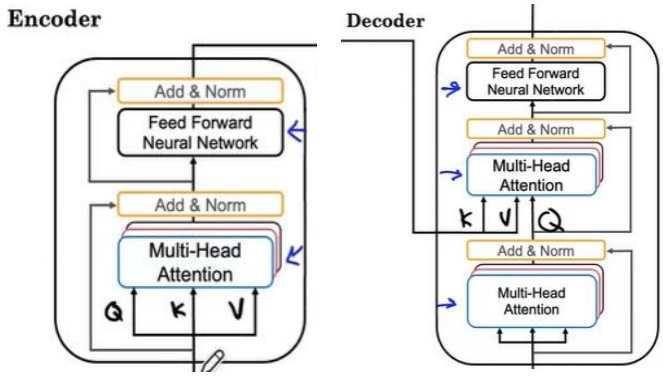
위치 인코딩이 sin 과 cos 을 사용하여 수행하는 작업은 고유한 위치 인코딩 벡터를 만든다.



→sin
→cos
→sin
→cos

pos = 1 인 첫 번째 단어의 p<1> != 세 번째 단어의 p<3> ; 다른 벡터

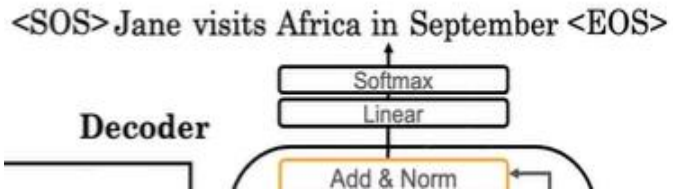
Positional Encoding 을 아키텍처에 포함시키기 위해 Residual Connections 를 이용해 더해 주고, 정규화 과정을 거친다. 그림에서는 Add & Norm 이라고 표현



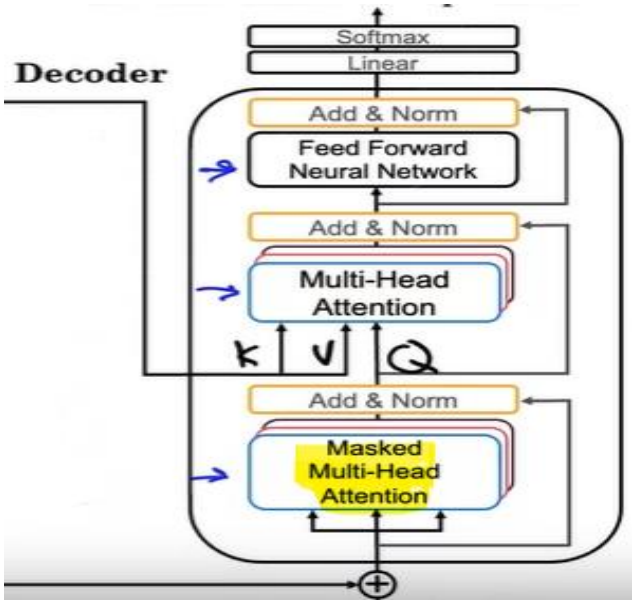
Bash Norm

(: 위치 정보를 위치 인코딩으로 전달) 레이어와, 유사한 layer Add & Norm 사용

: 학습 속도 빨라지고, 아키텍처 전체에서 반복



Decoder 출력에는 선형 계층과 softmax 계층이 있는데, 이 계층은 다음 단어를 한번에 한 단어씩 예측



Mask Multi-Head Attention

: Decoder 에서 사용되는데, 훈련할 때 데이터셋이 정확하다면, 처음부터 끝까지 확인할 필요가 없다는 점에서 용이하다. 정확한 문장이 주어지면, 뒷부분만 숨겨둔 뒤 앞의 정확한 몇 단어를 보았을 때 뒷부분을 어느 정도로 예측하는지를 확인하는 것이다.

** RNN, GRU, LSTM : 한번에 한 단어씩 정보 처리(순차 아키텍처)

Transformer Network : 모든 문장을 동시에 수집