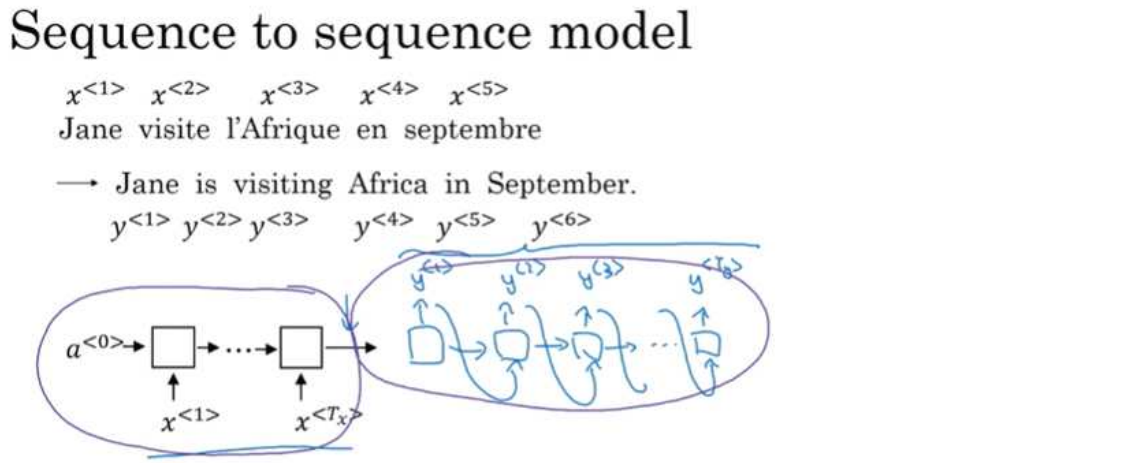
6주차 Sequence Model
Various sequence to sequence architectures

[Basic Models]
이번 강의부터는 Sequence-to-sequence 모델(Seq2Seq 모델)에 대해서 배우게 된다.
Basic model부터 시작해서 Beam search와 attention model에 대해서 알아보자.

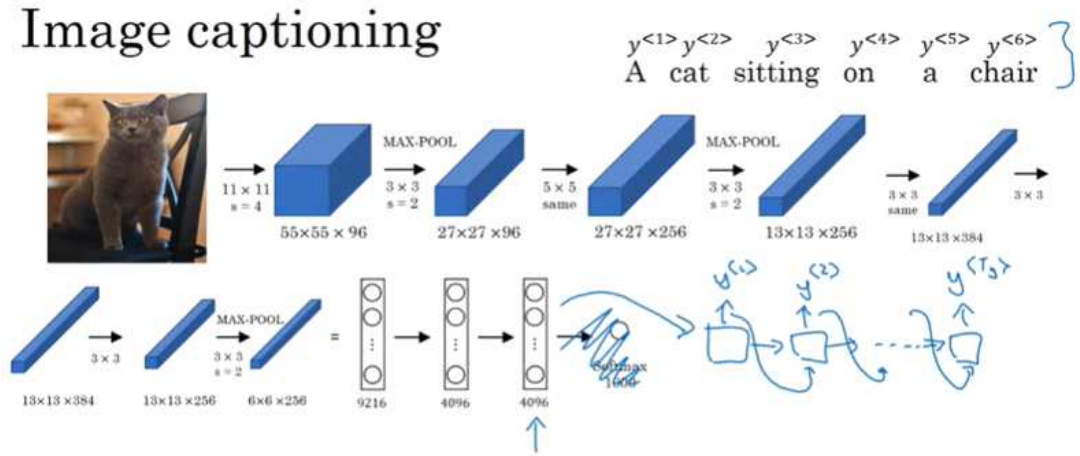
'Jane viste l'Afrique en septembre'라는 프랑스어로 된 문장을 영어 문장으로 변환하고 싶다면, 프랑스어로 된 문장 시퀀스를 $x^{<1>}$부터 $x^{<5>}$까지 표시하고, $y^{<1>}$ 에서 $y^{<6>}$까지 사용해서 output 시퀀스 단어를 표시한다. 그렇다면 어떻게 새로운 network를 학습해서 시퀀스 x를 입력으로 하고 시퀀스 y를 출력할 수 있을까?



[Sutskever et al., 2014. Sequence to sequence learning with neural networks]
[Cho et al., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation]
Andrew Ng

위와 같은 모델을 통해서 가능한데, 이 모델은 인코더(Encoder) 네트워크와 디코더(Decoder) 네트워크로 구성된다.
프랑스어 문장을 단어 단위로 인코딩하여(단어는 순차적으로 인코더 RNN에 들어감) 인코더 네트워크를 통해 하나의 고정된 벡터(컨텍스트 벡터)로 변환한다-> 인코더로부터 전달받은 벡터를 사용하여 디코더 네트워크는 영어 문장의 단어를 순차적으로 생성한다
이 모델은 충분한 프랑스어 문장과 영어 문장의 데이터셋이 있을 때, 효과가 있음

이와 비슷한 구조(Encoder-Decoder 구조)로 **Image captioning(이미지 캡션 생성)**도 수행 가능



Mao et. al., 2014. Deep captioning with multimodal recurrent neural networks]
Vinyals et. al., 2014. Show and tell: Neural image caption generator]
Karpathy and Li, 2015. Deep visual-semantic alignments for generating image descriptions]

Andrew Ng

1. 인코더 (이미지에서 특징 추출)

- CNN 활용: 인코더로는 이미지의 특징을 추출하는 합성곱 신경망(CNN)이 사용한다 – 이 예시에서는 pre-trained된 AlexNet을 사용하고, 마지막 softmax layer를 제거한 후에 AlexNet으로 4096차원의 특징벡터를 출력한다. 이는 이미지의 시각적 정보를 요약한 고차원 벡터이다.

2. 디코더 (텍스트 생성)

- RNN 활용: 디코더는 RNN 구조를 사용하며, CNN에서 추출된 4096차원의 특징 벡터를 입력으로 받아, 이를 바탕으로 이미지를 설명하는 단어를 순차적으로 생성한다. 번역 모델과 유사하게 진행되며, 단어 하나하나를 출력한다.
- 디코더 RNN은 첫 번째 단어 "A"를 생성한 후, 그 다음에는 이 단어와 함께 현재 상태를 활용하여 두 번째 단어 "cat"을 생성하는 방식으로 진행한다. 이렇게 RNN은 반복적으로 단어를 생성하여 이미지에 대한 설명을 완성한다.

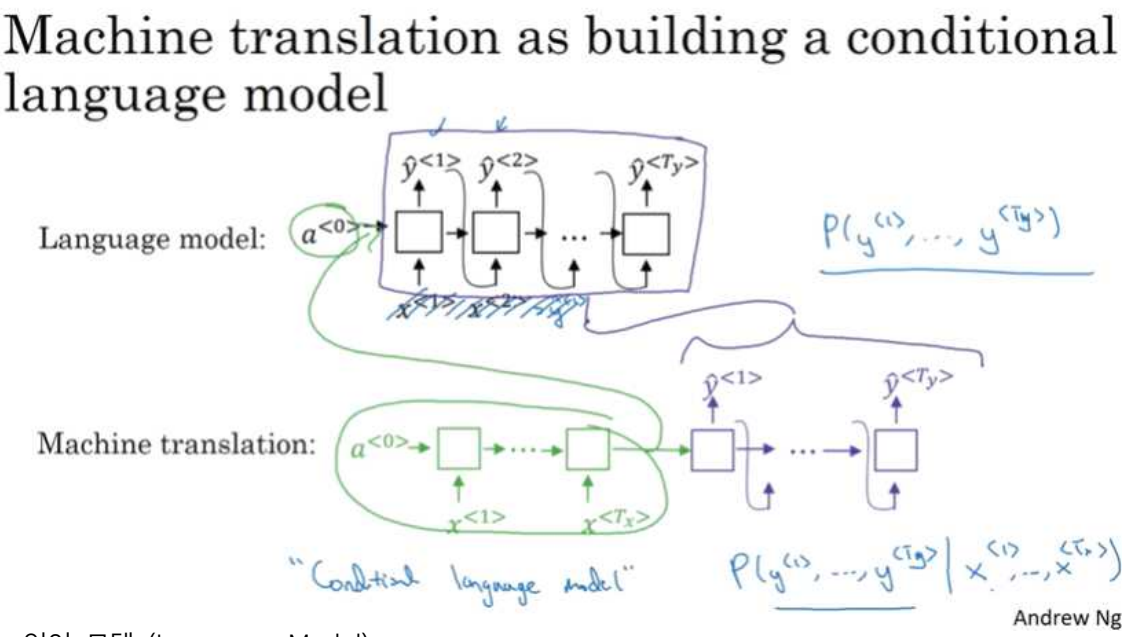
위에서 언급한 두 모델에서 우리는 무작위로 선택된 번역을 원하지 않거나, 가장 정확한 번역을 원할 수 있고, 이미지 캡션 또한, 무작위의 캡션을 원하지 않고, 가장 정확한 캡션을 원할 수 있다.
다음으로 어떻게 정확한 번역이나 캡션을 생성할 수 있는지 알아보도록 하자.

[Picking the most likely sentence]

Machine translation(기계번역) 모델과 강의의 첫 주에 배웠던 language 모델은 몇 가지 비슷한 점과 차이점이 있다.

기계번역은 conditional language model(조건부 언어모델)으로, 아래의 기계번역 모델의 구성을 살펴보면, 뒤쪽의 디코더 네트워크가 language 모델과 유사한 것을 볼 수 있다.

즉, language 모델은 0벡터에서 시작했지만, 기계번역 모델에서는 인코더 네트워크를 통과한 출력이 language 모델로 입력되는 것과 같다.

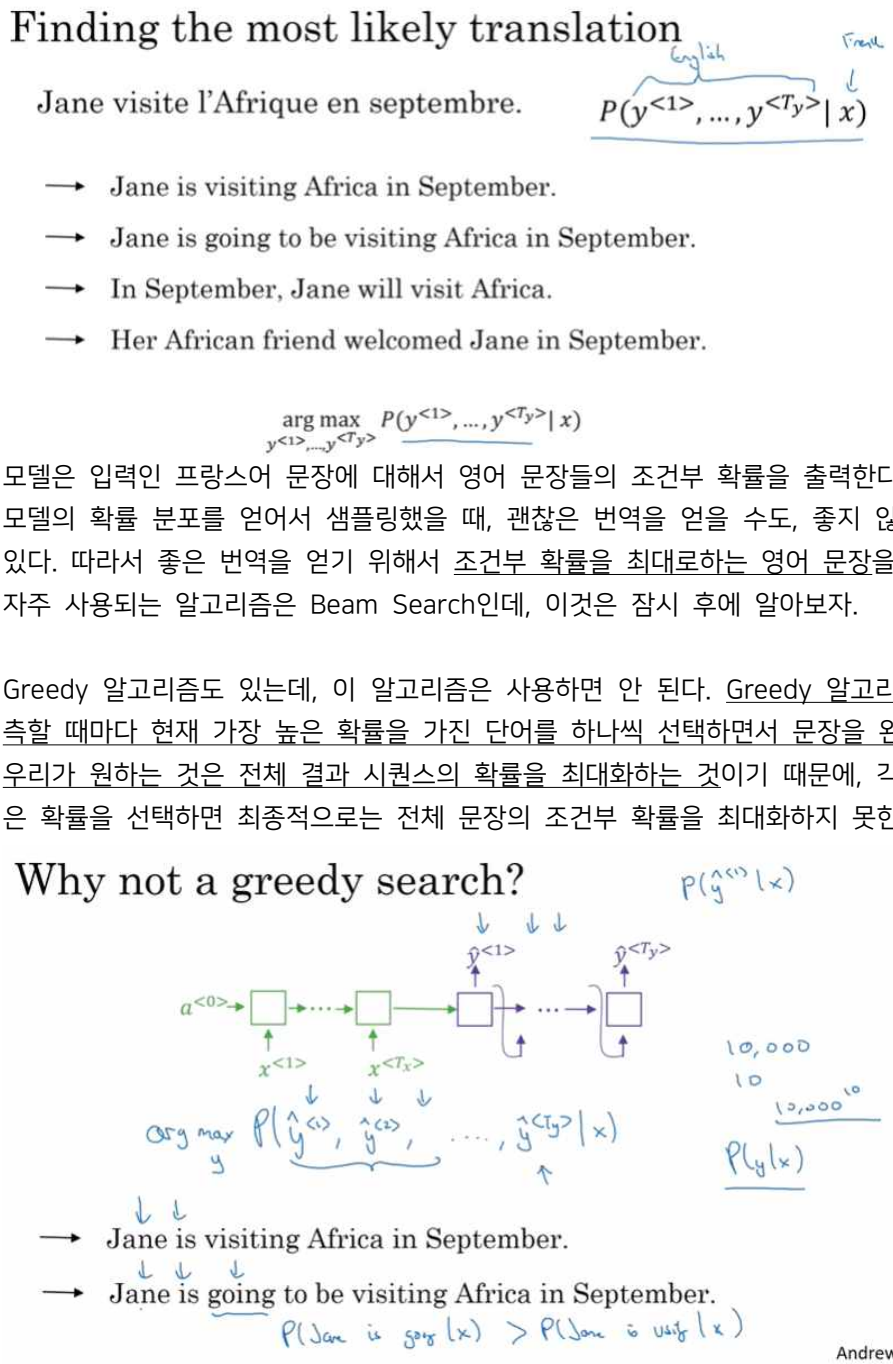


1. 언어 모델 (Language Model)

- 언어 모델은 주어진 단어 시퀀스에 이어 나올 단어의 확률을 예측한다. 예시에서는 입력 시퀀스가 없으며, 시작 토큰(<a0>)을 통해 문장을 생성하기 시작한다.
- 이 모델은 다음 단어의 확률을 이전 단어와 현재 상태를 기반으로 반복적으로 예측한다. 확률 분포는 $P(y(1),y(2),...,y(Ty\ ))$로 표현되며, 단어의 순서에 따라 문장 전체의 확률이 계산된다.

2. 기계 번역 (Machine Translation) 모델

- 기계 번역 모델은 조건부 언어 모델(Conditional Language Model)로 볼 수 있다. 여기서는 언어 모델과 달리 입력 시퀀스가 존재하며, 이 입력 시퀀스는 원본 언어(예: 프랑스어)의 문장이다.
- 입력 문장(x)은 인코더 네트워크를 통해 고정된 벡터로 변환되며, 이 벡터는 원본 문장의 전체적인 의미나 정보의 요약을 담고 있고, 디코더의 시작 상태로 전달된다. 이후 디코더는 이 컨텍스트 벡터를 기반으로 타겟 언어(예: 영어)의 문장을 생성한다. 이 과정을 통해 프랑스어 문장 x 가 주어졌을 때 영어 문장 y 를 생성하며, 그 확률은 $P(y^{<1>},\cdots,y^{<T_y>}|x^{<1>},\cdots,x^{<T_x>})$ 로 표현된다. 이 확률은 "이전 단어들 및 컨텍스트 벡터를 고려했을 때, 타겟 언어의 다음 단어가 될 가능성이 가장 높은 단어는 무엇인가"를 의미한다.-> 가장 높은 확률의 가장 적절한 번역을 찾는다



모델은 입력인 프랑스어 문장에 대해서 영어 문장들의 조건부 확률을 출력한다.
모델의 확률 분포를 얻어서 샘플링했을 때, 괜찮은 번역을 얻을 수도, 좋지 않은 번역을 얻을 수도 있다. 따라서 좋은 번역을 얻기 위해서 조건부 확률을 최대로하는 영어 문장을 찾아야 한다.
자주 사용되는 알고리즘은 Beam Search인데, 이것은 잠시 후에 알아보자.

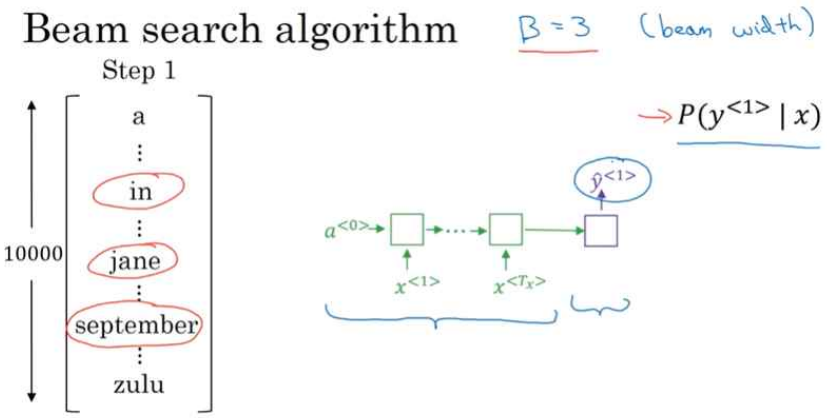
Greedy 알고리즘도 있는데, 이 알고리즘은 사용하면 안 된다. Greedy 알고리즘은 다음 단어를 예측할 때마다 현재 가장 높은 확률을 가진 단어를 하나씩 선택하면서 문장을 완성하는 방식이다.
우리가 원하는 것은 전체 결과 시퀀스의 확률을 최대화하는 것이기 때문에, 각 단계에서만 가장 높은 확률을 선택하면 최종적으로는 전체 문장의 조건부 확률을 최대화하지 못한다.

위 예시처럼, 'Jane is'까지 두 단어를 선택한 후에, 세번째 단어를 선택할 때를 살펴보자.
Jane is 다음에는 going이라는 단어가 더 흔하게 사용되기 때문에 greedy 알고리즘으로는 going이 선택될 확률이 높다. 하지만 전체 문맥을 고려하지 못하므로 잘못된 번역을 생성할 수 있다.

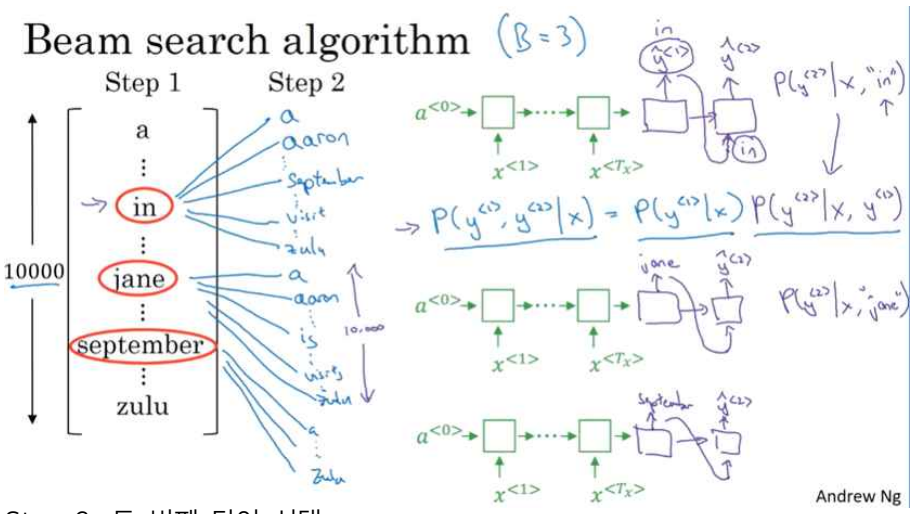
따라서 Greedy 알고리즘의 한계를 극복하기 위해서 전체 문장에 대한 확률을 최대화해야한다.
따라서 heuristics 탐색 알고리즘, 특히 Beam Search을 사용해서 전체 문맥을 어느 정도 고려하면서 최적의 단어 시퀀스를 찾는다. 항상 최대 확률의 결과를 보장하지는 않지만, 이정도로 충분할 수 있다.

[Beam Search]
기계 번역과 같은 자연어 처리 모델에서 가장 높은 확률의 output 시퀀스를 찾는 데 사용되는 알고리즘이다. Greedy Algorithm이 매 단계마다 가장 높은 확률의 단어를 하나만 선택한다면, Beam Search는 그보다 더 넓은 범위의 후보를 고려하여 보다 최적에 가까운 시퀀스를 찾아낸다.

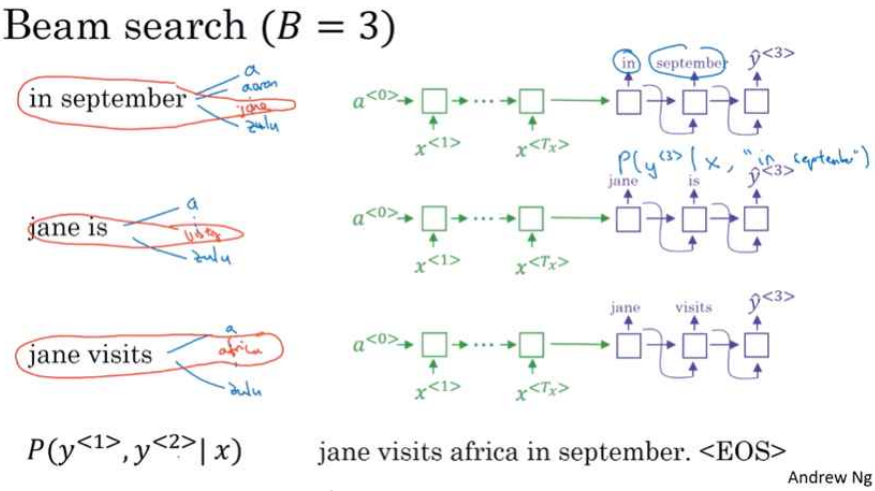
Beam Search가 어떻게 동작하는지 살펴보자.



Step 1: 첫 번째 단어 선택
1. Beam search는 Beam Width(B)라는 매개변수를 가지며, 이는 각 단계에서 고려할 후보의 수를 나타낸다. B=3면 매 단계에서 가장 높은 확률을 가지는 3개의 단어를 선택하여 탐색한다.
그래서 Step 1에서 가장 확률 $P(y^{<1>}|\ x)$이 높은 단어 3개 'in', 'jane', 'september'를 선택되었다. 이는 모델이 각각의 단어가 첫 번째로 올 확률이 가장 높다고 판단한 결과다.



Step 2: 두 번째 단어 선택
1. 두 번째 단어를 선택할 때는, Step 1에서 선택된 3개의 단어를 각각의 문맥으로 사용하여 다음에 올 수 있는 단어들의 확률을 계산한다.
2. 각 후보 시퀀스의 확률은 Step 1에서 선택된 3개의 단어의 조건부 확률로 계산되며, 식으로 표현하면 다음과 같다: $P(y^{<1>},y^{<2>}|\ x) = P(y^{<1>}|\ x)\ P(y^{<2>}|\ x,\ y^{<1>})$). 이는 첫 번째 단어와 두 번째 단어가 입력 x에 따라 함께 발생할 확률을 계산하는 것이다.
3. 이렇게 계산된 확률을 통해 가장 높은 확률을 가지는 상위 3개의 시퀀스가 선택된다. 예를 들어, "in September", "jane is", "jane visits"와 같은 시퀀스가 선택되었다.



Step 3: 세 번째 단어 및 이후의 단어 선택

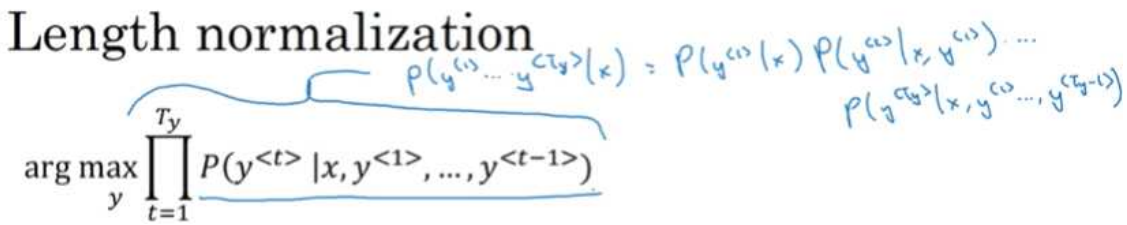
1. 세 번째 단어를 선택할 때도 같은 방식으로 진행되며, 현재까지의 시퀀스의 확률에 따라 가능한 후보 시퀀스를 확장해 나간다.
2. 각 단계에서 가장 높은 확률을 가진 3개의 시퀀스만 유지되며, 이렇게 계속 진행하여 최종적으로 종료 토큰 <EOS>를 만나면 종료하고, 가장 높은 확률의 시퀀스 1개, 영어 문장을 출력한다.

위 과정은 B=3일 때의 탐색 과정을 살펴본 것이고, 만약 B=1이라면 Greedy 알고리즘과 동일하다

다음으로 Beam search 모델이 더 정확하고 적절한 시퀀스를 생성하도록 최적화할 수 있는 몇 가지 팁에 대해서 알아보자.

[Refinements to Beam Search]

1. Length Normalization (길이 정규화)



Beam Search는 주어진 입력에 대한 번역의 조건부 확률을 최대화하는 것을 목표로 한다.
그리고 조건부 확률을 풀어서 쓰면 다음과 같다.

$$P(y^{<1>}, \cdots, y^{<T_y>} | x) = P(y^{<1>} | x) P(y^{<2>} | x, y^{<1>}) \cdots P(y^{<T_y>} | x, y^{<1>}, \cdots, y^{<T_y - 1>})$$

그러나 확률 P는 각 단어의 발생 확률을 곱하는 것이기 때문에, 시퀀스가 길어질수록 전체 확률은 기하급수적으로 작아진다(확률 P는 모든 값이 1보다 작음). 이렇게 확률이 너무 작아지면 모델이 짧은 시퀀스를 더 선호하는 문제가 발생한다. (short output 선호)

- **로그 변환을 통한 수치 안정화**: 이를 해결하기 위해 확률에 로그를 취하여 계산한다. 로그를 취하면 더 많은 단어를 포함하는 시퀀스라도 수치적으로 안정화되어 계산이 용이해진다.

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

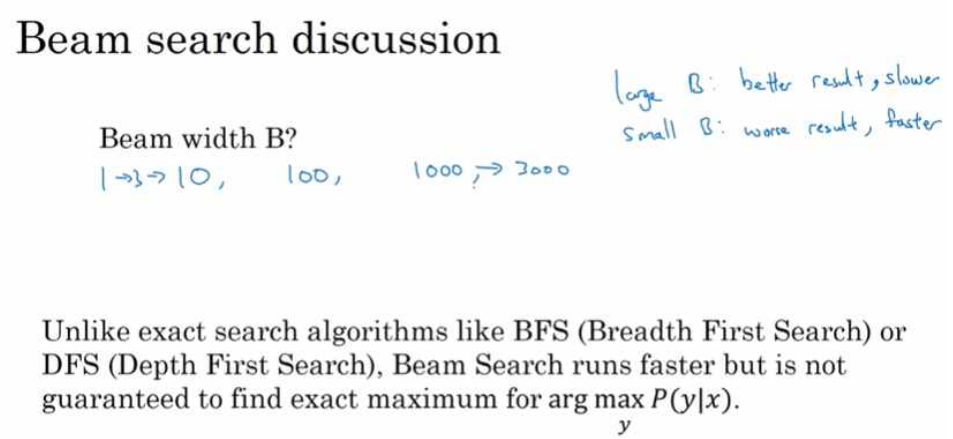
확률에 로그를 취해도 최대화하는 값은 변하지 않기 때문에 문제가 되지 않는다. 즉, 로그를 취하는 과정은 계산을 안정화하기 위한 수단이지 결과에 영향을 주지 않는다.

- **길이 정규화**: 그런데, 이 경우에도 확률은 항상 1보다 작거나 같기 때문에, 더 많은 단어가 있을수록 음수가 점점 커지게 된다(0과 1 사이의 값인 조건부확률에 로그를 취하면 음수 값이 됨. 시퀀스의 단어가 많아질수록 로그 확률을 더하므로 합산된 값은 더 작은 음수가 됨.). 이를 해결하기 위해 시퀀스의 길이 T_y로 normalization(정규화)한다.

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

하이퍼파라미터 α로 길이의 중요성을 조정할 수 있다. (α = 1: 완전한 정규화, α = 0: 정규화를 하지 않는 경우, 일반적으로 α = 0.7 등 중간값을 선택하여 최적의 결과를 얻는다.)

2. Beam Width B의 선택



Beam Width B는 Beam Search에서 매 단계마다 유지할 후보 시퀀스의 개수를 결정하는 중요한 파라미터이다.

- 큰 B: 많은 경우의 수를 고려하기 때문에 더 정확한 결과를 얻을 가능성이 높다. 그러나 연산량이 많아져 느려지고 메모리 사용량이 커진다.
- 작은 B: 고려하는 후보 시퀀스가 적어 빠르게 결과를 얻을 수 있지만 번역의 질이 저하될 가능성이 있다.
- 예시는 3개의 가능성을 살펴보았지만(B=3), 실제로 B는 10 정도로 작게 설정하기도 하고, 더 좋은 번역을 위해 1000에서 3000까지 크게 설정하기도 한다.

그리고 BFS나 DFS와 같은 탐색 알고리즘과 다르게, Beam 탐색은 훨씬 더 빠른 알고리즘이다. 그러나 정확한 최대값을 보장하지는 않는다. 이는 Beam Search가 근사적인 최적화 알고리즘이기 때문이다.

[Error analysis in beam search]
Beam Search는 휴리스틱 탐색 알고리즘이며, 항상 best 확률의 문장을 출력하지는 않는다.
이때, 만약 Beam Search에서 mistake가 있으면 어떻게 해야 할까?

Error analysis은 번역 모델의 정확도를 향상시키기 위해, 오류가 Beam Search 알고리즘에 의한 것인지, 아니면 RNN 모델 자체의 문제인지를 구분하는 과정이다.

프랑스어 문장을 인간이 번역한 것(y★)(이상적)과 알고리즘이 Beam Search을 통해 번역한 결과물(^y)이 있다.

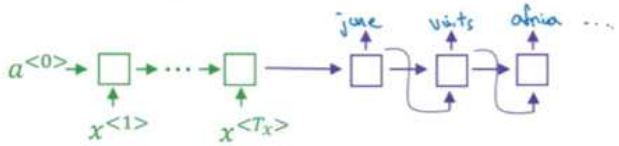
Example

Jane visite l'Afrique en septembre.

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

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

RNN computes $P(y^*|x) > P(\hat{y}|x)$



두 번역의 차이를 분석하기 위해 두 문장의 조건부 확률 P(y★| x)와 P(^y| x)을 비교한다.

Error analysis on beam search

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

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

Case 1: $P(y^*|x) > P(\hat{y}|x)$

Beam search chose $\hat{y}$. But y^* attains higher $P(y|x)$.

Conclusion: Beam search is at fault.

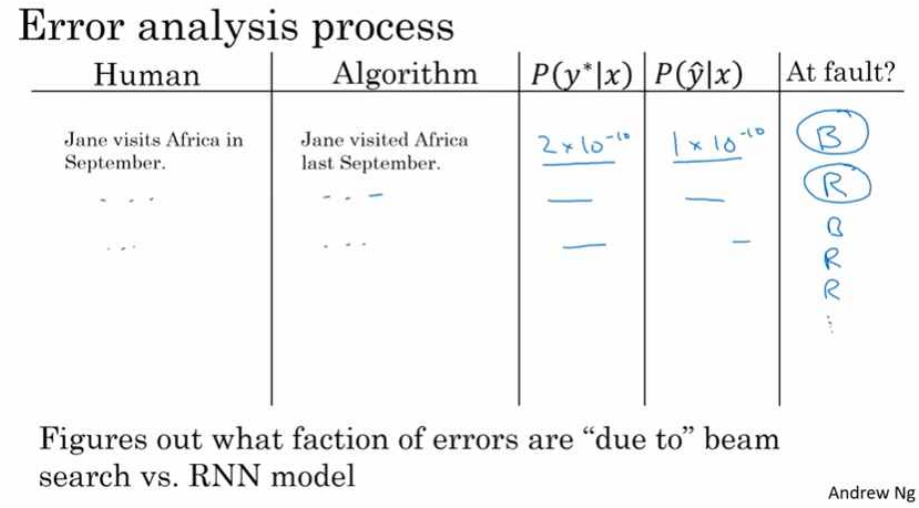
Case 2: $P(y^*|x) \leq P(\hat{y}|x)$

y^* is a better translation than $\hat{y}$. But RNN predicted $P(y^*|x) < P(\hat{y}|x)$.

Conclusion: RNN model is at fault.

Case1: P(y★| x)> P(^y| x): Beam Search가 최적의 확률을 찾지 못했다는 뜻. Beam Search 알고리즘의 문제라고 판단한다.
Case2: P(y★| x)<= P(^y| x): 알고리즘의 번역 ^y에 대한 확률이 더 높다면, RNN 모델 자체가 잘못된 번역을 더 높은 확률로 예측했기 때문이다. RNN 모델에 문제가 있다고 판단한다.

그리고 아래 과정을 통해서 RNN의 모델과 Beam Search 알고리즘에 의한 오류의 수를 파악할 수 있다. 이런 Error analysis process를 통해서 효율적으로 개발의 방향을 설정할 수 있다.



[Bleu(Blingual Evaluation Understudy) Score]

기계번역에서의 challenge 중의 하나는 꽤 괜찮은 번역을 여러 개 만들 수 있다는 것이다. 하나의 정답이 있는 이미지 인식과는 달리 여러 가지의 좋은 정답이 있다면 기계번역 시스템은 어떻게 이것들을 평가할 수 있을까? 정확성을 측정하면 되는데, 보편적으로 Bleu Score라는 방법이 정확성을 평가하는 대표적인 지표로 사용된다.

사람이 번역한 참조 번역(reference)과 비교하여 기계가 번역한 결과물이 얼마나 잘 번역되었는지 평가하는데 활용된다. BLEU 점수가 높을수록 기계 번역이 참조 번역과 유사하다는 의미이다.

1. 기본 아이디어

BLEU Score는 번역된 문장이 참조 번역과 얼마나 일치하는지 측정한다. 모델이 예측한 결과가 사람이 제공한 reference와 가깝다면 Bleu Score는 높다. 참조 번역이 여러 개일 수도 있다.

예: 프랑스어 문장 'Le chat est sur le tapis'의 번역에 대해 두 개의 참조 번역이 있을 때:

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

2. Precision과 Modified Precision

Bleu Score는 직관적으로 기계가 생성하는 글의 유형을 인간이 만들어낸 reference에서 나타나는지 살펴보는 방법이다. 극단적인 기계번역의 결과를 가지고 어떻게 점수가 계산되는지 살펴보자.

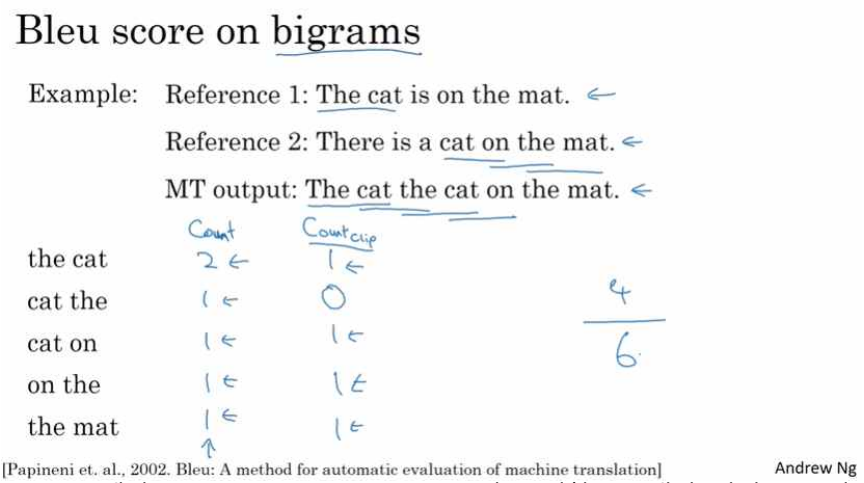
- Precision: 기계 번역된 문장의 각 단어가 참조 번역에서 나타나는지 확인하여 계산한다.
  예를 들어, the the the the the the the라는 번역이 있다면, 이 단어는 Ref1이나 Ref2에 모두 나타나므로 정밀도는 7/7=1.0이 된다. 하지만 이 결과는 문장의 정확성과는 무관하게 항상 높을 수 있다.
- Modified Precision: 정밀도의 문제(기계 번역이 과도한 반복을 통해 높은 점수를 얻는 것)를 해결하기 위해, 참조 번역에 나타난 단어의 최대 빈도를 고려한다.
  각 단어에 대해서 중복을 제거하고, reference 문장에 나타난 최대 횟수만큼 점수를 부여한다. Ref1에서는 'the'가 2번 나타나고, Ref2에서는 'the'가 1번 나타난다. 따라서, Modified precision은 2/7이다.

3. n-grams을 이용한 BLEU Score

지금까지 우리는 각 단어를 개별적(isolated word)으로 살펴보았다. 즉, 단어의 순서를 고려하지 않았다. BLEU Score는 문맥과 단어의 순서를 고려하기 위해 n-grams(단어 묶음)을 사용한다.

단어들의 순서와 조합을 고려하면서 번역이 얼마나 원문과 비슷한지 평가할 수 있다.

- Unigrams: 단일 단어를 대상으로 계산한다.
- Bigrams: 두 개의 연속된 단어 쌍을 기준으로 계산한다.



bigrams에서 Bleu score는 MT output의 근접한 두 개의 단어를 묶어서, Reference에 얼마나 나타나는지 체크해서 Modificed Precision을 계산한다.

The cat", "cat the", "the cat", 등이 각각의 n-grams이 참조 번역에 얼마나 등장하는지 계산하여 Modified Precision을 구한다.

4. BLEU Score 계산

BLEU Score는 다양한 n-grams을 고려한 정밀도를 조합하여 계산한다.

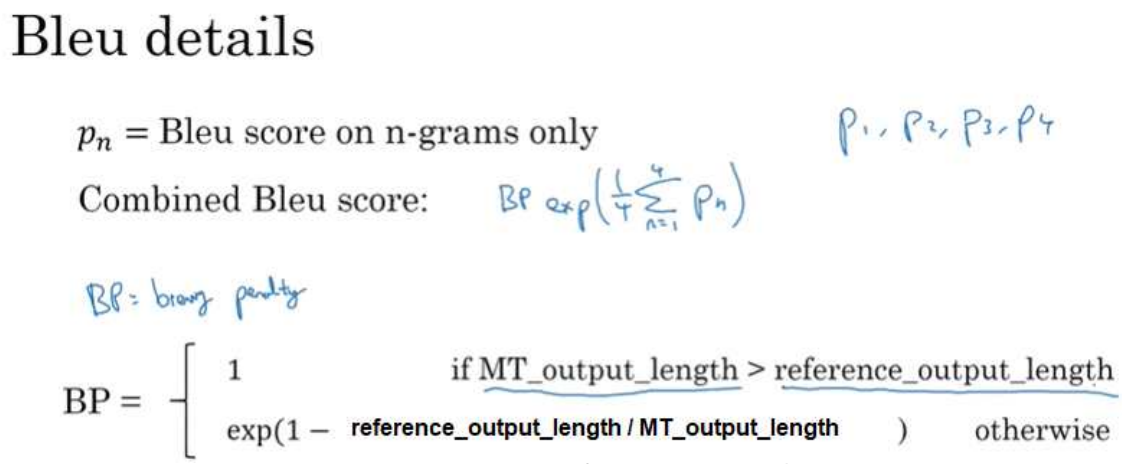
P_n: 각 n-grams(1, 2, 3, 4-grams)의 BLEU Score이다.

최종 combined BLEU Score는 아래와 같이 계산된다:

$$BP \cdot \exp\left(\frac{1}{4}\sum_{n=1}^{4} \log P_n\right)$$

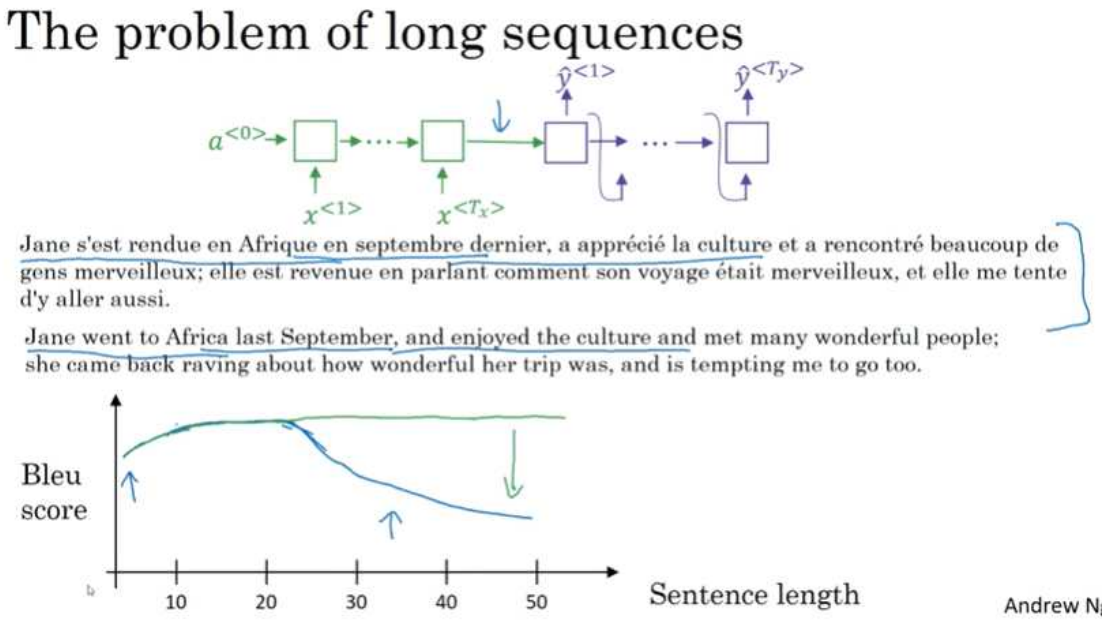
n-gram을 사용하여 번역 결과와 참조 번역 간의 일치율을 측정하며, 여러 n-gram의 평균을 계산하여 최종 점수를 산출한다.

여기서 BP(Brevity Penalty)는 번역된 문장이 너무 짧은 경우 패널티를 부여하는 요소다.



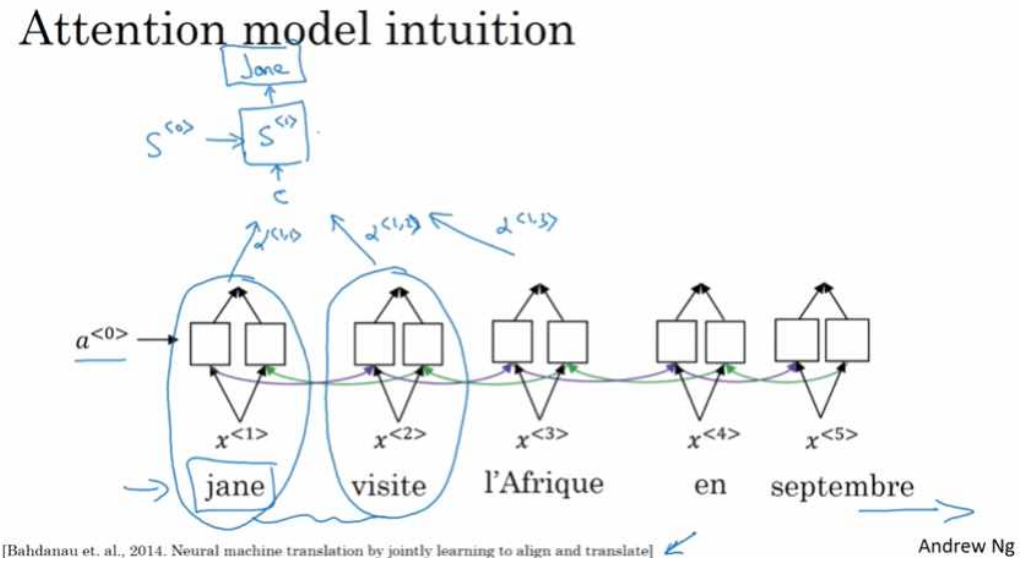
Bleu Score는 완벽하지는 않지만 수치적으로 평가할 수 있는 꽤 괜찮은 Metric이다.

[Attention Model]

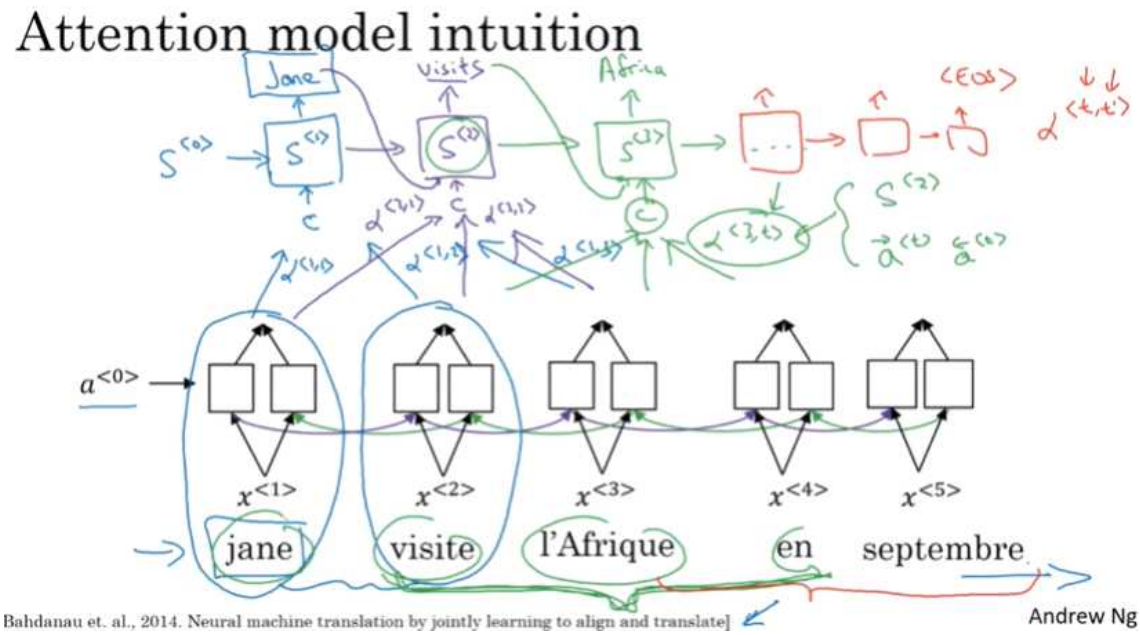


- Attention Model은 기존의 Encoder-Decoder 구조(대부분의 기계번역에서 사용)에서 긴 문장의 번역 문제를 해결하기 위해 고안되었다. Encoder-Decoder 구조는 일반적으로 짧은 문장에서 잘 동작하지만 입력 문장의 길이가 길어질수록 Context Vector가 모든 정보를 담기 어려워 성능이 낮아지면서 Bleu score가 낮아진다(파란색 그래프). 여기에 Attention model을 사용하면 긴 문장에서도 성능을 유지할 수 있게 된다(초록색 그래프).
- 사람이 번역할 때, 문장 전체를 외워서 처음부터 번역하는 것이 아닌 중간 중간 번역하는 것처럼 Attention model은 사람과 유사하게 번역한다.
- 기본적으로 Attention Model은 번역 과정에서 전체 문장이 아닌 특정 부분에 집중하여 효과적으로 번역하는 방법을 도입한다. 예측할 단어와 연관되는 부분만 집중해서('Attention') 참조한다.

Attention model은 처음에 기계번역 용도로 개발되었지만, 여러 App 영역에서도 적용되고 있다. 그렇다면 Attention model이 어떤 방식으로 동작하는지 간단하게 살펴보자.



- Encoder 부분은 RNN(또는 BRNN)으로 구성되며, 입력 문장의 각 단어를 처리하면서 Hidden State(사진의 각 사각형 노드)를 생성한다.
- Decoder 부분은 각 시점의 예측을 수행하는데, 이때 디코더의 Hidden State(activation) $s^{<t>}$와 Context Vector (C)를 활용한다.
- Hidden State는 디코더가 이전에 생성한 단어와 문맥 정보를 바탕으로 다음 단어를 예측하는 데 필요한 정보를 담고 있다.
- Context Vector (C)는 번역할 단어와 관련된 정보만을 선택적으로 반영한 벡터다. C를 생성할 때, Encoder에서 생성된 각 시점의 Hidden State 중 필요한 정보에만 집중해서 C를 계산한다.
- Attention Weights를 통해 어떤 부분을 집중해서 참고할지 결정한다. 각 단어의 예측 시점마다 입력 문장의 특정 부분에 집중(Attention)하여 그에 따른 C를 만들어내고, 이를 통해 번역을 진행한다.
- 이러한 Attention Mechanism은 단어의 순서와 연관된 문맥 정보를 활용하는 데 효과적이어서, 기존의 Encoder-Decoder 구조가 가진 긴 문장 처리의 한계를 극복할 수 있게 해준다.



[Attention Model]

- Attention model은 이름에서 알 수 있듯이 디코더가 입력 문장의 특정 부분에 집중(Attention)하도록 도와준다.
-

Encoder와 Decoder의 구성:

- Encoder는 RNN 또는 BRNN(Bidirectional RNN)으로 구성되며, 각 입력 단어를 처리하면서 hidden state를 생성한다.
- 이 hidden state(activation)는 $\overrightarrow{a}^{<t'>}, \overleftarrow{a}^{<t'>}$ 로 표기된다.(t'은 input x의 time step을 의미함)
- Context Vector c는 이러한 hidden state 중 일부를 참조해 계산되며, 해당 디코더 시점의 예측에 필요한 정보를 제공한다.

Attention Parameter:

- 그리고 Decoder의 입력으로 사용되는 C는 아래와 같이 계산된다.(ex, 첫 번째 Decoder output)

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

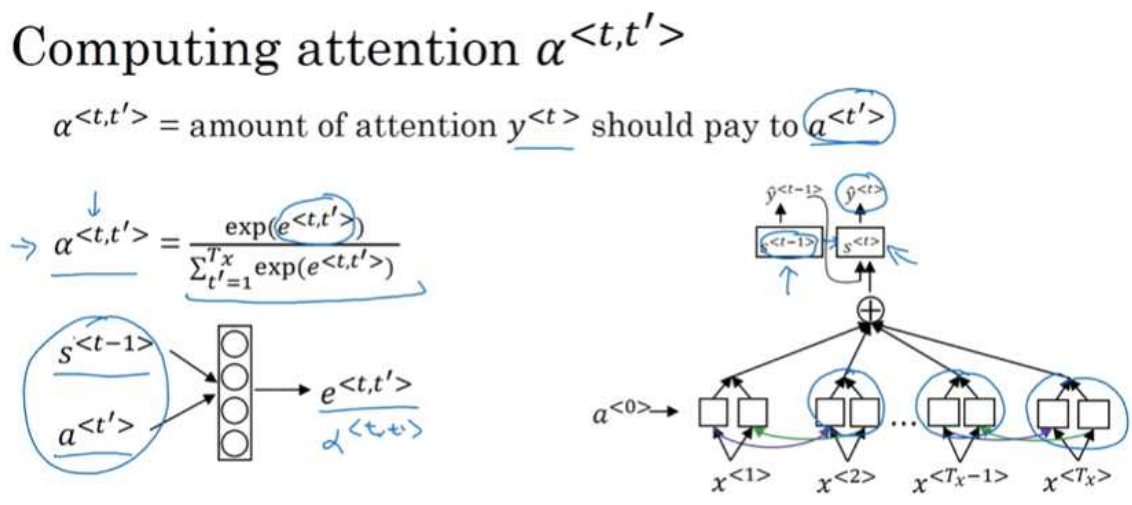
- 여기서 α는 Attention parameter로써 Context(C)가 activation과 feature에 얼마나 의존적인지를 나타내며, $y^{<t>}$예측에 얼마나 'attention' 해야 할 지를 나타내는 정도(amount)이다.
- $\sum_{t'} \alpha^{<1,t'>}$=1의 특징을 가지고 있으며, 왜 총합이 1이되는 지는 바로 뒤에서 설명하도록 한다.
- 두 번째 단어도 동일한 방법으로 계산된다.

$$c^{<2>} = \sum_{t'} \alpha^{<2,t'>} a^{<t'>}$$

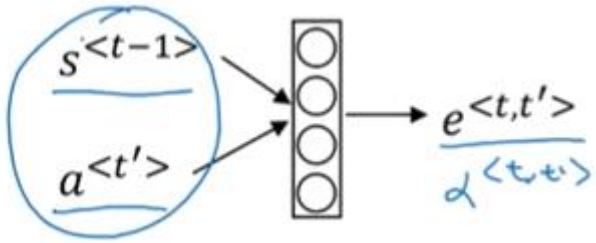
Attention Parameter 계산 방식:

attention parameter인 $\alpha<t,t'>$는 softmax를 통해 계산되며, 이를 통해 합계가 1이 된다.

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

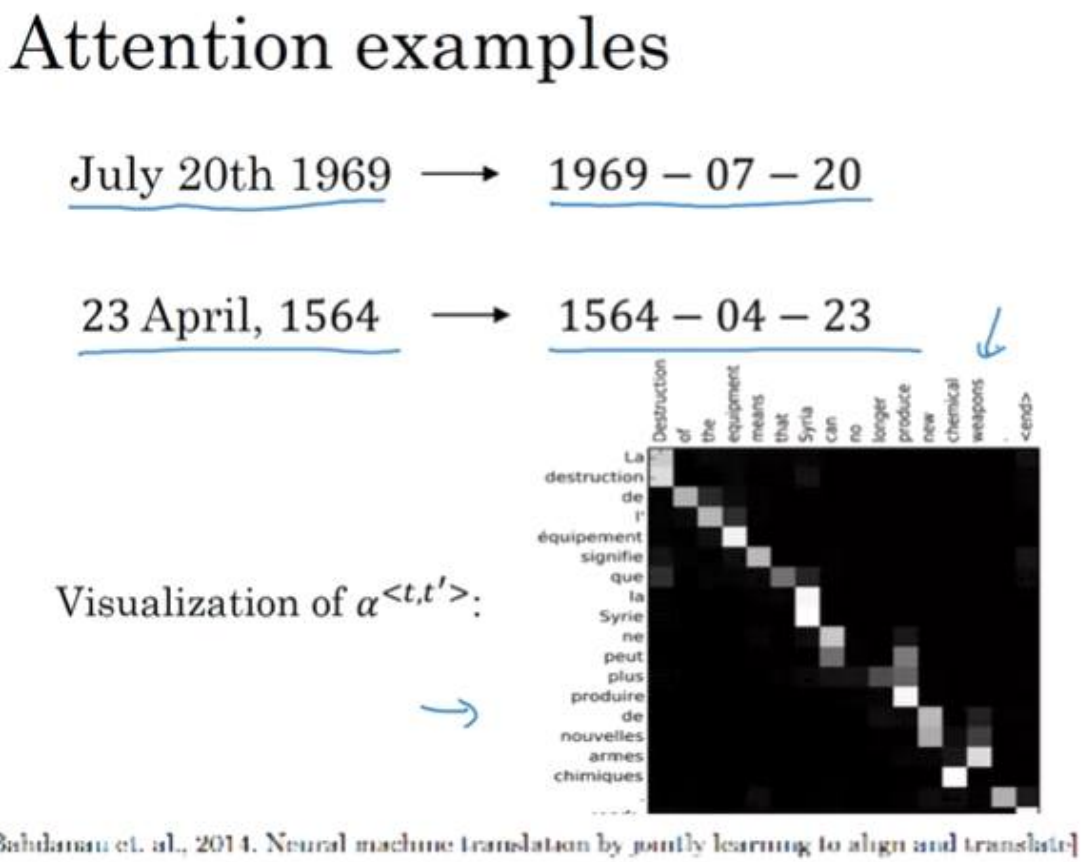


이때 $e^{<t,t'>}$은 작은 신경망(Dense layer)을 통해서 구할 수 있으며, 이전 layer의 hidden activation $s^{<t-1>}$과 $a^{<t'>}$을 입력으로 한다. (현재 단계에서 구하려고 하는 activation이 $s^{<t>}$이기 때문에 이전 layer의 hidden activation을 사용)



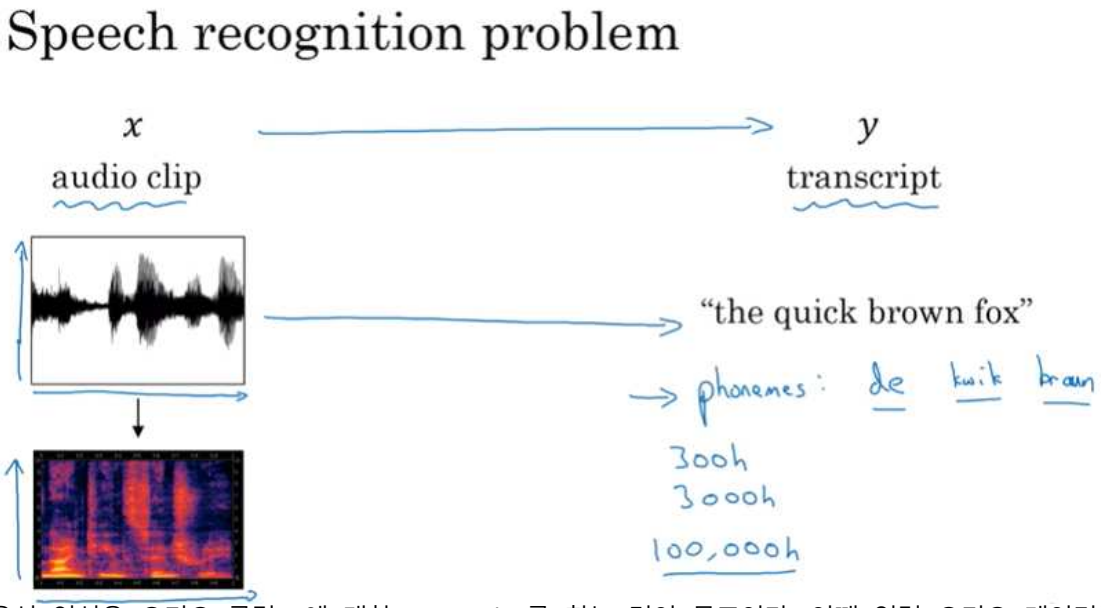
Attention 모델의 시간 복잡도:
Attention model의 단점은 학습 시간 복잡도가 이차(quadratic)라는 점이다. (학습시간(연산량)이 quadratic time(cost)을 가짐) 만약 입력 문장 길이 T_x와 출력 문장 길이 T_y가 있을 때, attention 파라미터의 총 개수는 $T_x \times T_y$이다. (이러한 cost를 절감하기 위한 논문도 존재한다.)

응용:
Attention model은 기계 번역뿐만 아니라 이미지 캡셔닝 등 다양한 영역에서 적용되고 있다
Attention example로 date normalization도 있다.



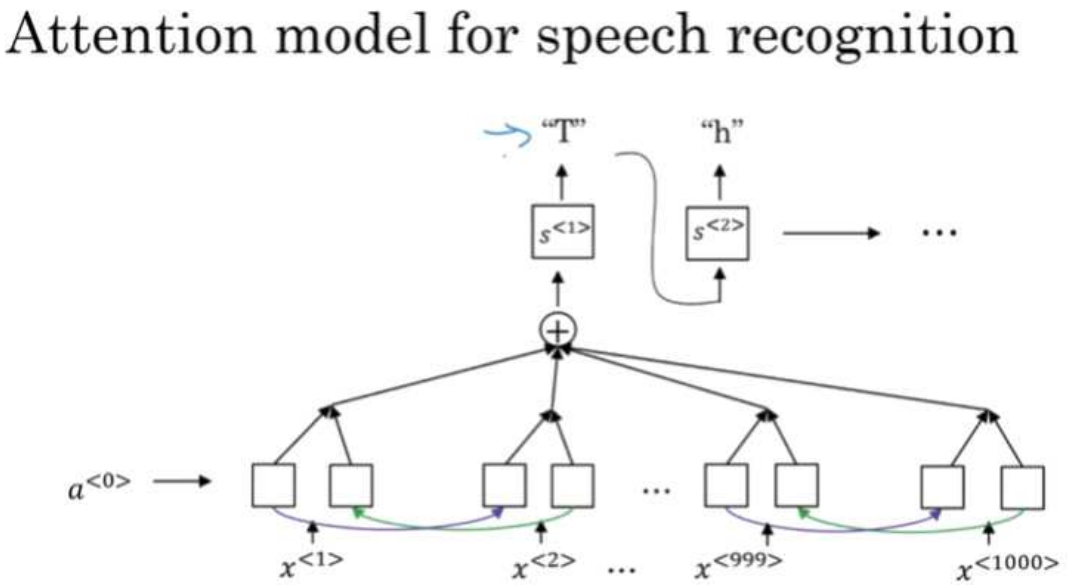
[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

[Speech recognition]



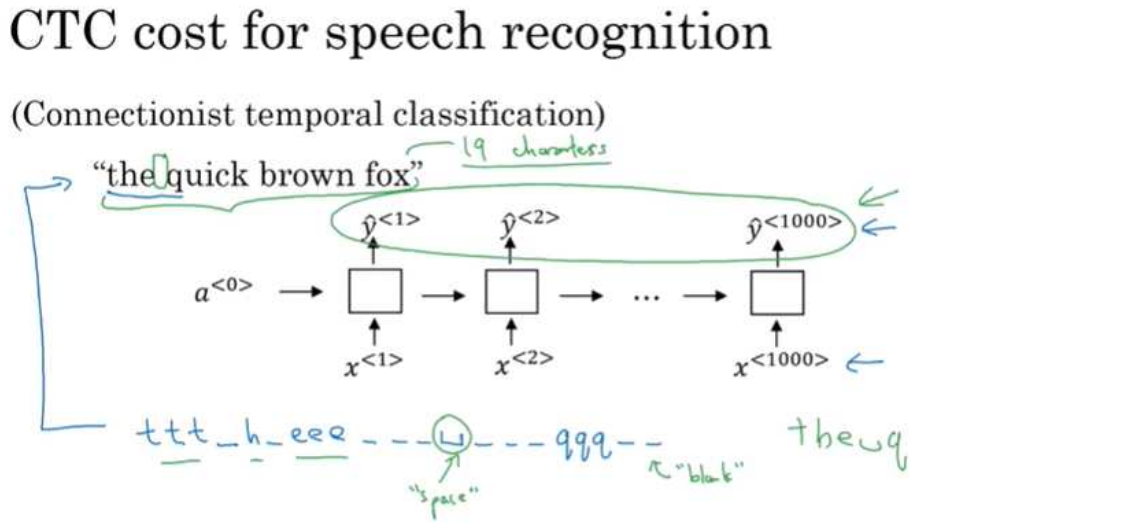
음성 인식은 오디오 클립 x에 대한 transcript를 찾는 것이 목표이다. 이때 입력 오디오 데이터를 주파수별로 분리하여 분석하는 스펙트로그램 같은 방법을 통해서 음성 신호를 시각화하고 처리한다. 오디오 클립의 그래프의 x축은 시간이고, y축은 주파수이다. 일반적으로 이렇게 전처리를 적용하는 경우가 많다.

그래서 어떻게 음성 인식 시스템을 구현할 수 있을까?



Attention model을 음성 인식 모델에 사용할 수 있다. Attention 모델은 입력의 각 부분에 집중할 수 있도록 설계되어, 입력 오디오의 시간 단계에 따라 단어를 예측한다. 이처럼 각 시점에서의 입력 데이터를 활용하여 텍스트를 생성하는 방식이 RNN 구조를 통해 이루어진다.

또 다른 방법으로는 음성 인식을 위해서 CTC Cost를 사용하는 것이다.



CTC는 시퀀스 데이터를 다루는 RNN과 함께 음성 인식, 필기 인식 등의 문제를 해결할 때 사용하는 손실 함수의 일종이다. 특히, 시퀀스의 길이가 입력과 출력이 다를 때 유용한 방법이다(긴 오디오를 짧은 텍스트로 변환). CTC의 목표는 입력 시퀀스 x (예: 오디오 신호)와 출력 시퀀스 y (예: 텍스트)의 정렬을 도와주는 것이다.

예를 들어, 'the quick brown fox'와 같은 문장을 오디오에서 추출할 때, 각 문자 사이에 소리가 안나는 부분(공백)에 빈칸을 추가한다. 즉, 블랭크('_')를 포함시킨다. 이를 통해 반복된 문자를 압축하여 하나의 문자로 취급한다. 즉, ttt_h_eee를 'the'로 처리하는 것처럼 말이다.

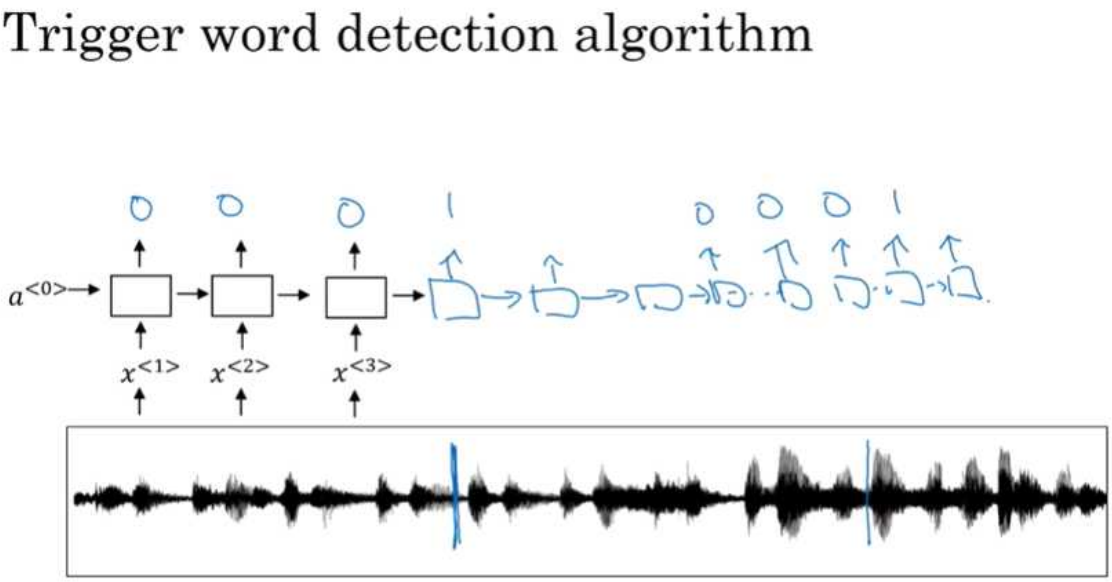
CTC의 규칙은: 1. 반복된 문자를 'blank'로 구분하지 않을 때 병합한다.

2. 최종 출력은 공백('_')을 최종 출력에서 제거하여 필요한 문자만 남겨서 구성된다.

이러한 모델과 손실 함수를 활용하여 음성 데이터를 정확하게 인식하고 전사할 수 있게 된다.

[Trigger Word Detection]

Trigger word detection은 기계를 깨울 수 있는 방법을 의미한다. 음성 비서(예: "Alexa", "Hey Siri", "Okay Google")처럼 특정 단어나 구가 인식되면 그 이후부터 음성 명령을 받아들이는 기능이다. 이를 위해 RNN 기반 모델을 사용하여 실시간으로 음성을 처리하고, 특정 "Trigger Word"가 말해지면 이를 탐지하는 알고리즘을 개발한다.



1. RNN을 통한 실시간 입력 처리:
음성 데이터(x)가 실시간으로 입력되면서 RNN이 이를 시퀀스 데이터로 처리한다. 시퀀스의 각 시점에서 모델은 해당 시점에서 모델이 오디오 데이터를 보고 이 순간이 Trigger Word와 연관되어 있는지에 대한 확률을 계산한다.

2. 라벨링 및 탐지:
RNN의 각 시점에서 "Trigger Word"가 인식될지 여부를 예측한다.
음성 데이터의 각 구간에 대해 '0' 또는 '1'로 라벨링하는데, '0'은 "Trigger Word"가 인식되지 않는 구간이고, '1'은 "Trigger Word"가 인식된 구간이다. 예를 들어 "Hey Siri"라는 Trigger Word가 말해지면, 그 구간 이후의 음성은 '1'로 라벨링된다.

3. Trigger Word 출력 및 알림:
"Trigger Word"가 감지되면 그 시점 이후의 예측값이 연속적으로 '1'이 된다. 그 결과로, "Trigger Word"가 인식되었음을 시스템에 알리게 되며, 이후 사용자의 명령을 처리하는 모드로 전환된다.

+) Trigger Word Detection에서는 '0'이 훨씬 많고 '1'이 적은 불균형 데이터가 생성된다. 이를 해결하기 위해 Trigger Word가 감지되었을 때 그 주변 구간을 '1'로 길게 설정해 모델이 더 쉽고 정확하게 탐지할 수 있도록 한다.
하지만, 이 경우에는 실제로 잘 동작하지 않는다. 왜냐하면, label의 불균형하게 분포하고 있기 때문인데, 1에 비해서 0이 훨씬 더 많다. 그래서, 단 한번이 아닌 label 1의 비율을 늘려서 출력하도록 할 수 있다.

서술형 문제 1:
Attention Mechanism이 기존의 Encoder-Decoder 구조를 개선한 방식에 대해 설명하시오. 또한, Attention Mechanism이 번역 성능을 향상시키는 이유를 서술하시오.
답:
Attention Mechanism은 기존의 Encoder-Decoder 구조에서 긴 문장의 번역 성능이 떨어지는 문제를 해결하기 위해 도입된 기술이다. 일반적인 Encoder-Decoder 구조에서는 입력 문장을 인코딩한 후, 하나의 고정된 Context Vector로 디코더에 전달한다. 긴 문장의 경우 Context Vector가 모든 정보를 담기 어려워 번역 성능이 저하되는 문제가 있었다.
Attention Mechanism은 번역할 각 단어 시점마다 필요한 인코더의 Hidden State만 집중해서 참고할 수 있도록 한다. 즉, 디코더의 각 시점에서 필요한 부분에 "Attention"을 주어, 해당 시점에 가장 중요한 정보를 효율적으로 사용한다. 이로써 긴 문장에서도 특정 부분에 집중하며 번역이 가능해져 번역 성능이 향상된다.
서술형 문제 2:
BLEU Score가 기계 번역의 품질을 평가하는 방법에 대해 설명하시오. 또한, BLEU Score 계산 시 Modified Precision을 사용하는 이유를 서술하시오.
답:
BLEU (Bilingual Evaluation Understudy) Score는 기계 번역의 결과물을 사람이 만든 참조 번역(reference)과 비교하여 번역 품질을 평가하는 지표이다. 일반적으로 n-gram을 사용하여 번역 결과와 참조 번역 간의 일치율을 측정하며, 여러 n-gram의 평균을 계산하여 최종 점수를 산출한다.
BLEU Score는 Modified Precision을 사용하는데, 이는 기계 번역이 과도한 반복을 통해 높은 점수를 얻는 것을 방지하기 위함이다. 단어의 등장 횟수를 제한하여 참조 번역에 나타난 빈도만큼만 점수를 부여한다. 예를 들어, 번역 결과에 "the the the"와 같이 중복된 단어가 나타나더라도 참조 번역에 "the"가 한 번 나타났다면 그 빈도를 기준으로 점수를 부여한다.
서술형 문제 3:
CTC (Connectionist Temporal Classification)의 목적과 동작 방식을 설명하시오. CTC가 음성 인식에 사용되는 이유를 포함하여 답변하시오.
답:
CTC (Connectionist Temporal Classification)는 입력과 출력 간의 길이가 다르고 정렬이 불명확한 시퀀스 데이터에 대한 학습을 위해 사용되는 손실 함수이다. 음성 인식에서는 입력 오디오 시퀀스와 출력 텍스트 시퀀스의 길이가 달라 정렬하기 어려운 문제를 다루는데, CTC는 이 문제를 해결한다.
CTC는 각 시점의 입력에 대해 출력의 모든 가능한 alignment를 고려하며, 출력 시퀀스 내에서 같은 문자가 연속해서 나올 경우 이를 한 번으로 간주하는 규칙을 따른다. 예를 들어, "heello"라는 입력이 있을 때, 연속된 "e"는 하나의 "e"로 간주한다. 또한, 각 시점의 입력에서 "blank" 토큰을 두어 중복을 피하고, 이 "blank"를 통해 실제 단어와 간격을 조정한다.
이러한 방식으로 음성 시퀀스와 문자 시퀀스를 매핑하면서도 정확한 타임스탬프 없이 전체적인 출력 텍스트를 예측할 수 있게 되어 음성 인식에서 유용하게 사용된다.