Lecture 08. Machine Translation, Sequence-to-sequence and Attention

Ⅰ. Pre-Neural Machine Translation

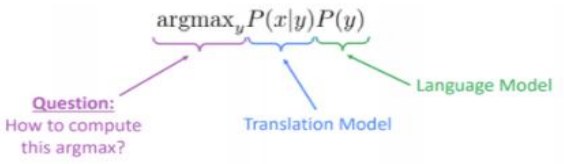
Machine Translation : 기계를 통해 번역을 하는 문제.

1. 1950's : Early Machine Translation

   냉전 시기에 소련과 미국이 항상 서로의 통신이나 기밀 문서들을 자동으로 번역해서 해석하기를 원하면서 등장. 주로 rule-based 방식(두 개의 언어의 사전을 통해 각 단어가 대응하는 것을 찾아서 번역하는 가장 단순한 방식)

2. 1990s-2010s : Statistical Machine Translation

   기존의 rule-based 방식이 아닌 data를 통해서 번역하는 방식이 처음 도입됨. 통계적 기계번역이라고 불리고, 데이터를 통해 확률 분포를 학습하는 방법이다. 특정 언어 input sentence x에 대해서 또 다른 언어 output sentence인 y로 번역할 때 다음의 확률이 가장 높은 sentence를 선택한다. 실제로 사용할 때는 Bayes Theorem을 사용.



   앞부분은 Translation Model로 parallel 데이터를 통해서 확률을 계산하고 뒷부분은 Language Model로 monolingual 데이터를 통해서 확률을 계산한다. 일종의 분업 형태인데 왼쪽은 작은 단어나 구를 번역하는 역할을 하고 오른쪽은 좋은 문장이나 좋은 구조를 뽑아내는 역할을 한다. 왼쪽 translation model은 병렬데이터, 즉 문장들이 두 개의 언어 모두로 표현되어 있는 데이터로 한 문장에 대해 두 가지 언어 표현을 통해 한 언어를 다른 언어로 바꿨을 때의 확률을 계산할 수 있게 한다. 그리고 오른쪽의 monolingual 언어 데이터는 하나의 언어만 있는 데이터로 언어의 특성을 확률 값으로 계산한다. Translation model P(x|y)를 학습하기 위해서는 많은 양의 parallel data가 필요하다. Translation model을 학습시킬 때 필요한 방법이 alignment(정렬).

3. Alignment

   병렬처리된 문장에서 특정 단어쌍들의 대응 관계. 정렬에서 가장 큰 문제는 두 개의 언어는 각각 특성이 달라서 단어들의 순서나 품사의 순서가 다르고 1대1대응이 어려울 수 있음. 정렬이 many to one 즉, 여러 단어가 하나의 단어로 정렬될 수도 있다. 특히, many-to-many로 많은 단어가 많은 단어로 정렬되는 계속 혼잡한 상황이 발생하고 1대1로 잘

정렬이 되지 않는 경우가 문장에서 많이 존재한다. 이 상황에서 정렬을 학습시킬 수 있는 방법은 아래와 같다.

(1) 무차별 대입

가능한 모든 y를 열거해서 모든 확률을 계산해 본다. 하지만 굉장히 오래 걸리며 비용이 많이 소모될 수 있음.

(2) Heuristic 알고리즘

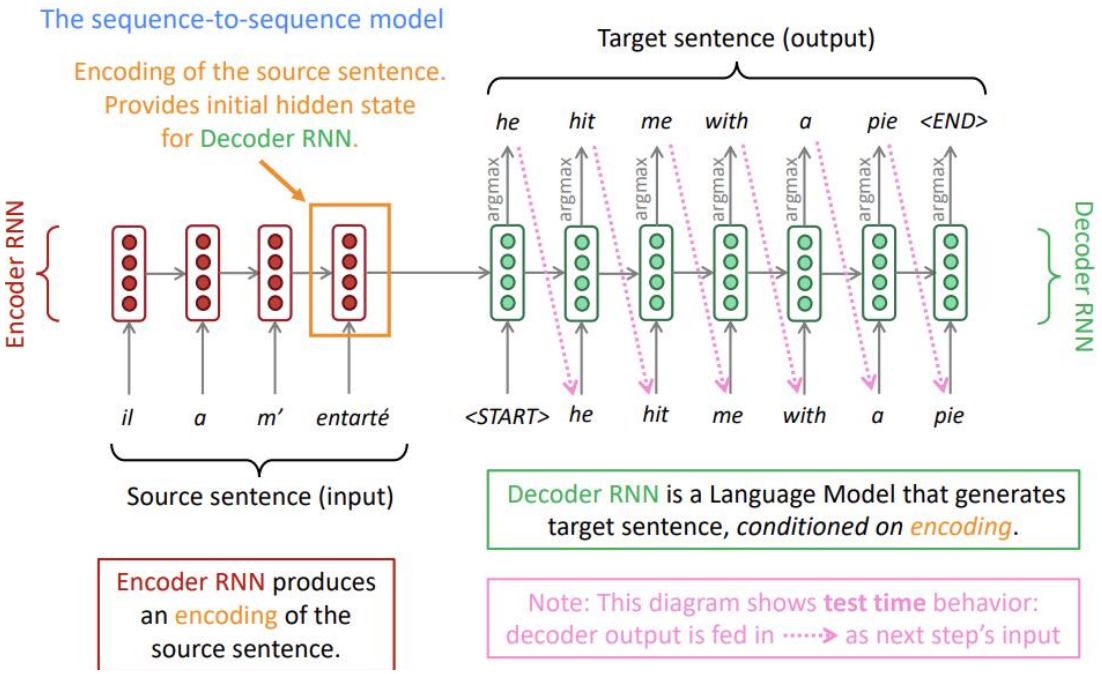
너무 낮은 확률들은 계산하지 않고 높은 확률만 계산하고 다른 것들은 버리는 방식.

다양한 가설들을 세우고 나무가 가지를 치는 것처럼 너무 낮은 확률은 버리면서 높은 확률로 가는 형식 ->이 과정이 decoding!

통계적 기계 번역은 좋은 성능을 내지만 매우 복잡한 구조이고 각 system은 각 부분으로 나눠서 sub-system들이 모여 있는 형태이다. 많은 feature engineering과 추가적인 많은 자료가 필요하며 사람의 손을 많이 거쳐야 한다.

Ⅱ. Neural Machine Translation(Sequence to Sequence)

위에서의 한계점을 극복하게 된 방법이 '신경 기계 번역(Sequence to Sequence)'이다.



단일 신경망으로 기계번역을 하는 방법이다. 두 개의 RNN(인코더 RNN, 디코더 RNN)으로 이루어져 있고 한 시퀀스를 다른 시퀀스로 매핑한다. Encoder에서는 문장을 마지막에 encoding시켜 decoder rnn에 넘겨준다. Rnn은 단방향, 양방향, lstm 등 모두 될 수 있다. 디코더 rnn은 조건부 언

어 모델이라고 하는데, 인코딩에 따라 첫번째 출력을 얻고 다음에 나올 단어의 확률 분포 argmax를 취해준다. 다음 단계에서 디코더에 전 단계의 단어를 넣고 다시 피드백하여 argmax를 취하고 단어를 맞춘다. 이런 방법을 계속 반복하여 목표 문장을 찾는다. 디코더가 종료 토큰을 생성하면 중단한다. 이 모델은 기존 기계번역보다 아래와 같은 특징 때문에 훨씬 더 유용하다.

- 긴 텍스트를 짧은 텍스트로 요약이 가능하다.
- 전의 단어와 다음 단어로 이어지며 문맥적인 것을 파악할 수 있다.
- Input text를 output parse로 순서에 따라 parsing할 수 있다.
- 자연어를 코드로 생성할 수 있다.

이 모델은 조건부 언어 모델임. 디코더가 다음 타깃을 계측하는 조건부 언어 모델로 확률을 직접 계산한다. x가 주어지면 y를 뽑아내고 그 첫번째 y를 x와 다시 input으로 넣고 계속 반복하며 conditioning한다. 이 모델은 병렬 말뭉치를 학습을 한다.

NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots\underbrace{P(y_T|y_1,\ldots,y_{T-1},x)}_{\text{Probability of next target word, given target words so far and source sentence } x}$$

학습을 하는 방법은 시스템을 전반적으로 최적화하는 것이다. 인코더 RNN에 문장을 넣고 디코더 RNN의 모든 단계에서 다음에 올 확률분포의 손실을 교차 엔트로피 등을 통해 계산하고 얻은 손실을 더해 단계 수로 나누어 최종 loss값을 구한다. 역전파는 전체 시스템에 걸쳐 흐른다. 개별적으로 학습하는 것은 좋지 않고 end-to-end 방식으로 학습하는 것이 문장 요소를 모두 고려해 좋은 결과물을 도출한다. 미리 정의된 최대 길이에서 최대한 짧은 문장을 채우는 방법으로 진행된다.

1. Greedy decoding

   가장 확률이 높은 단어를 각 단계에서 도출하다 보니 결정을 수정할 수 없는 문제가 발생한다. 아래는 이를 해결하는 방법이다.

   (1) Exhaustive search decoding

$$\begin{aligned} P(y|x) &= P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x) \\ &= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x) \end{aligned}$$

철저하게 가능한 모든 언어 번역 공간을 검색하는 방법이다. 모든 개별 확률 분포의 곱으로 나타내며 디코더의 각 단계 t에서 가능한 부분 번역의 거듭제곱으로 v를 취적해야 한다. (V는 어휘의 크기, v는 복잡도의 지수) 모든 경우의 수를 다 계산하기 때문에 v가 너무 커서 굉장히 비용이 많이 든다.

(2) Beam search decoding

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

일종의 검색 알고리즘으로 디코더의 각 단계에서 k개의 가장 가능성이 높은 부분 번역을 선택하여 가지치기 하는 방식으로 진행된다. K는 빔의 크기로 k를 증가시킬수록 각 단계에서 더 많은 것을 고려하게 된다. k개의 가장 가능성이 높은 번역, 가설의 점수가 가장 높은 번역을 것이 핵심이다. 이 방법이 최적의 솔루션을 찾아주는 것은 보장하지 못하지만, 전부다 계산하는 것보다는 훨씬 효율적이다.

Greedy decoding과 beam search decoding을 비교해보면, greedy decoding은 모델이 end token을 생성할 때까지 decode를 진행하지만 beam search decoding은 한 가설이 end token을 생성하고 끝이 나면 그것은 따로 두고, 다시 다른 가설들을 탐색하기 시작한다. Beam search도 무한정 가설을 세울 수 없기 때문에 한계를 정해주어야 하는데 미리 정한 시간 단계 t에 도달하면 멈추는 방법, n개의 완료된 가설을 만들면 중지하는 방법이 있다. 이렇게 완성된 가설 모음 중 가장 높은 점수를 받은 것을 선택한다. 문제는 긴 가설일수록 낮은 점수를 받게 되어서 기준이 애매하다는 것이다. 가설이 길수록 -값이 계속 더해지기 때문에 어쩔 수 없이 정확하지 않더라도 짧은 가설을 채택하는 경우가 생긴다. 이를 해결하기 위해서는 간단하게 time step 수로 나누어 평균을 구하여 최상위를 선택하면 된다.

2. 장점

-기존 기계번역보다 sequence를 고려하기 때문에 좀 더 유차하고 문맥을 고려하여 단어와 구 사이의 유사성을 잘 이용하기 때문에 더 나은 성능을 제공한다.

-single neural network가 처음부터 끝까지 전반적인 시스템을 가지고 최적화된다. 개별적으로 최적화되어야 하는 기계번역과 달리 역전파가 전 시스템을 거쳐서 진행되기 때문에 단순하고 편리하다.

-인간 공학적 노력이 덜 필요하다. Feature engineering이 필요가 없으며 모든 언어쌍에 같은 방법으로 동일한 아키텍쳐로 적용이 가능하므로 병렬 말뭉치만 찾으면 된다.

3. 단점

-어디서 오류가 발생했는지 오류를 표시하는 것이 어렵다. 기계번역은 디버그가 가능하지 만 NMT는 쭉 흘러가는 형태이기 때문이다.

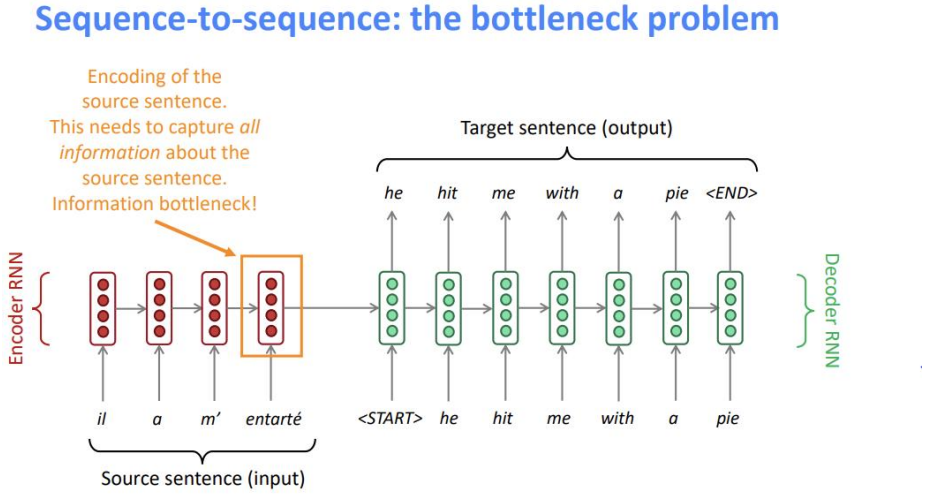
-특정 오류를 발견했을 때 이를 수정하고 제어할 사후규칙을 만들기 어렵다. 기계번역은 그 부분만 따로 떼어와서 제어하게끔 만들 수 있는데 NMT는 FLOW가 전반적으로 흘러가기 때문에 특정 부분만을 수정하기 어렵다.

4. How do we evaluate Machine Translation?

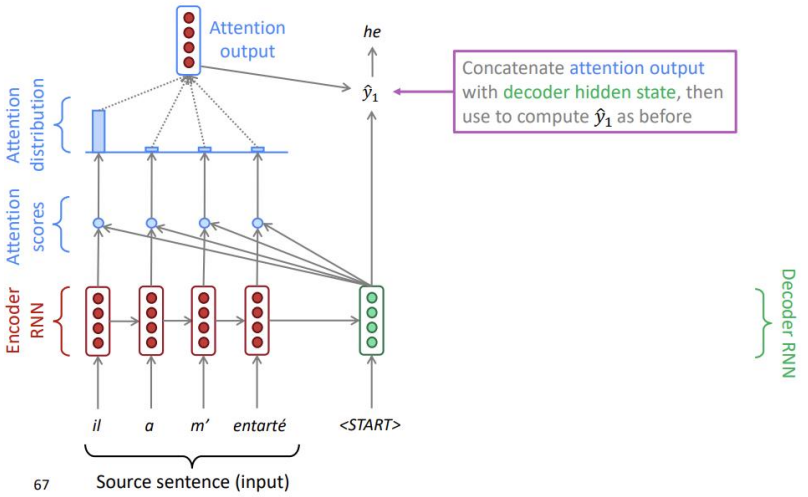
BLUE(Bilingual Evaluation Understudy) : 기계에 의해 번역된 문장과 사람이 작성한 문장을 비교해서 유사도를 측정해서 성능을 측정하는 지표.

기존 SMT 기계번역은 3년 동안 추세의 기울기가 완만했지만 NMT의 성능이 굉장히 빠르게 증가하였다. NMT는 현재까지도 연구가 진행되고 있다. NMT에서 기계번역의 모든 한계점이 없어진 것은 아니다. 단어의 범위를 초과하면 그 목표 단어를 만들어 낼 수 없다는 점(학습을 시킬 때 없었던 단어는 만들어 낼 수 없다.), 학습데이터와 test데이터 사이에 domain이 일치하지 않으면 성능이 떨어진다는 점, 한 문장을 encoding하는 것이기 때문에 긴 text의 경우 context를 계속 유지하여 오류가 생길 수 있다는 점, 리소스가 부족한 언어 쌍의 경우, 성능이 떨어질 수 있다는 문제가 있다. 또한, 관습적인 언어와 상식에 대한 개념이 없고 굉장히 편향되게 번역할 수 있다는 것이다.(paper jam) 또한 단순하게 반복한 문장을 멋대로 해석하는 문제가 발생하기도 하는데 이 sequence-to-sequence의 한계점을 보완하기 위해 나온 방법이 Attention이다.

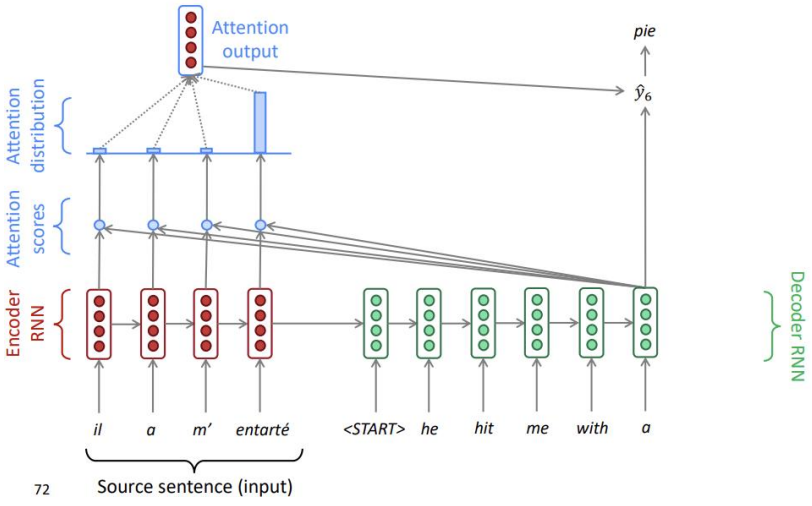
Ⅲ. Attention



Sequence-to-Sequence에서는 소스 문장의 인코딩이 중요하다. 여기서 문제는 단일 벡터 전체가 소스 문장 전체의 인코딩이라는 것이며, 문장의 정보가 끝에서 다 인코딩이 되어버리는 즉, 정보가 쏠려버리는 병목현상이 나타나게 된다. 소스 문장에 대한 모든 정보를 끝에서 다 캡처하도록 강조하며, 여기서 너무 많은 압력이 강해진다. Attention은 단계별로 집중을 하게 만들어 한꺼번에 집중이 가해지는 이 병목현상을 해결해준다. 디코더의 각 단계에서 인코더에 직접 연결을 하여 소스 시퀀스의 특정 부분마다 집중을 하게 만드는 작업을 한다.



디코더의 첫 단계에서 인코더의 첫 단계와 내적을 시켜서 스칼라 값을 뽑아 attention score를 구한다. Attention score란 현재 디코더의 시점 t에서 단어를 예측하기 위해 인코더의 모든 은닉 상태 각각이 디코더의 현시점의 은닉 상태 states와 얼마나 유사한지를 판단하는 점수이다. 그렇게 4개의 attention score를 가져와서 softmax에 넣고 확률분포를 얻는다. Bartchart로 알아본 결과, 대부분의 확률 질량이 첫번째 단어로 갔음을 알 수 있으므로 먼저 he라는 단어를 만든다. 이렇게 얻은 attention 분포는 인코더의 hidden states의 가중치가 된다. Attention 분포와 인코더의 hidden states를 가중 합하여 attention output을 뽑고 그 attention output과 디코더의 hidden state를 결합하여 y1이라는 결과 he를 도출하게 된다.



Attention 출력을 사용하여 다음 단어에 영향을 주고 그런 식으로 두번째 디코더에서도, 세번째 디코더에서도 같은 것을 반복한다. 각 단어에는 분포가 있으며, 기존 NMT보다 훨씬 더 유연하고 부드러운 정렬과 사고를 하게 된다. 기종 인코딩 방법으로는 마지막에 압력을 가해서 정보의 병목현상이 일어났는데, attention을 사용하여 각 인코딩의 각 단계마다 집중을 해주어 정보의 흐름을 마지막에 집중시키지 않고 각 단계마다 부드럽게 풀어준다. 식으로 나타내면 아래와 같다.

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

73

1. Attention의 효과

(1) 기존 NMT의 성능을 훨씬 향상시켰다. Decoder가 소스 문장 부분마다 집중을 하였기 때문으로 보인다.

(2) 병목현상을 해결했다.

(3) 기울기 소실 문제를 해결했다. 문장이 길어질수록 기울기 소실 문제가 발생했는데 attention은 각 단계마다 집중을 할 수 있어 해결한 것으로 보인다.

(4) NMT는 flow가 전반적으로 진행되기 때문에 어디서 오류가 발생했는지 추적하기 어렵지만 attention은 encoding의 각 단계마다 집중하여 보면서 어디서 오류가 발생했는지 확인할 수 있다.