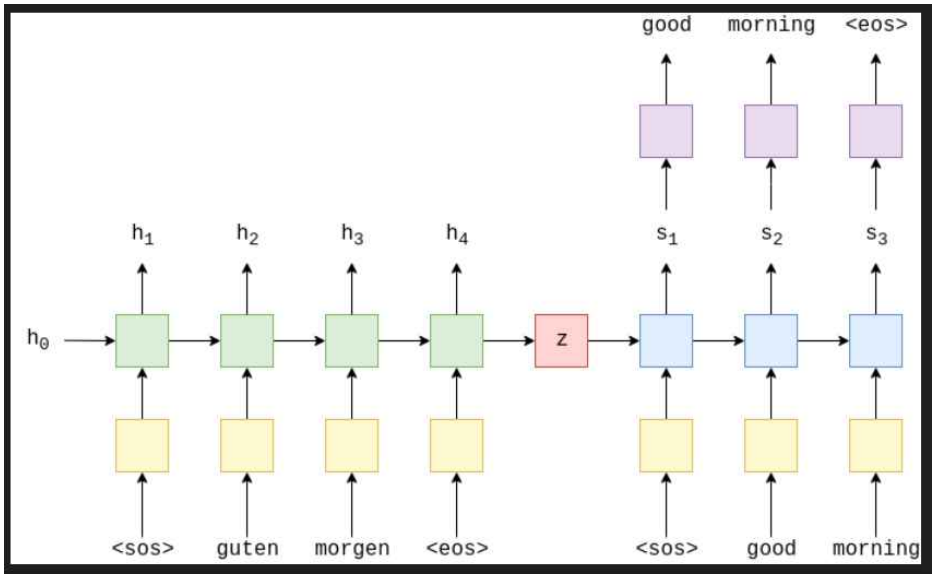
NLP Assignment01

v2022116 임영윤

1 - Sequence to Sequence Learning with Neural Networks

DNN은 음성 인식, 사물 인식 등에서 꾸준한 성과를 내어왔다. 하지만 input size가 fixed 된다는 한계점이 존재하기 때문에 sequential problem을 제대로 해결할 수 없다는 한계점이 존재했다. 이 논문에서는 2개의 LSTM을 각각 Encoder, Decoder로 사용해 sequential problem을 해결하고자 했다. 이를 통해 많은 성능 향상을 이루어냈으며, 특히나 long sentence에서 더 큰 상승 폭을 보였다. 이에 더해 단어를 역순으로 배치하는 방식으로 성능을 향상시켰다.

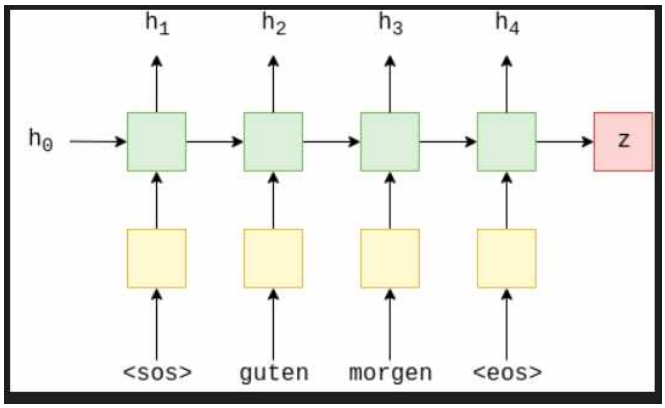


RNN은 사실 sequential problem에 매우 적절한 model이다. 하지만 input size와 output size가 다른 경우에 대해서는 좋은 성능을 보일 수 없었다. 본 논문에서 제시하는 model은 Encoder LSTM에서 하나의 context vector를 생성한 뒤 Decoder LSTM에서 context vector를 이용해 output sentence를 생성하는 방식으로 RNN의 한계점을 극복하고자 했다. input과 output sentence 간의 mapping을 하는 것이 아닌, input sentence를 통해 encoder에서 context vector를 생성하고, 이를 활용해 decoder에서 output sentence를 만들어내는 것이다. Encoder LSTM의 output인 context vector는 Encoder의 마지막 layer에서 나온 output이다. 이를 Decoder LSTM의 첫 번째 layer의 input으로 넣게 된다. 여기서 주목할만한 점은 input sentence에서의 word order를 reverse해 사용했다는 것이다. 또한 EOS token을 각 sentence의 끝에 추가해 variable length sentence를 다루었다.

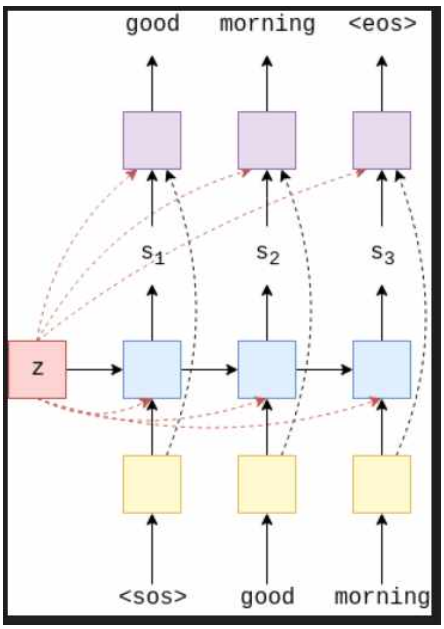
2 - Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation

이 논문은 RNN(GRU)를 이용한 Encoder Decoder 방식을 이용한 Translation Model을 제시함으로써 NMT방식을 제안하고 있다. 다만 이 논문은 NMT라는 용어를 사용하지 않고, 조심스럽게 SMT 이외의 다른 방식이 존재하고, 이러한 방식이 충분히 가능성이 있다는 approach를 보여준다. 이는 SMT방식의 process 중 일부를 대체하는 방식에서 확인할 수 있다.

이 논문에서 제안하는 NMT방식은 seq2seq방식이다. 다만 우리가 흔히 알고 있는 구조와는 다른 방식을 보여준다. 우리가 seq2seq방식이라 하면 Encoding Sequence의 마지막 output EOS와 마지막 Cell State를 Input으로 받아서 Decoder가 처리해주는 방식으로 보여주게 되어 위의 방식보다는 조금 더 많은 정보를 처리하는 것을 볼 수 있다.



Encoder 구조는 흔히 알려져 있는 seq2seq 방식과 크게 차이가 없다.



Decoder 방식은 다르다.

3 - Neural Machine Translation by Jointly Learning to Align and Translate

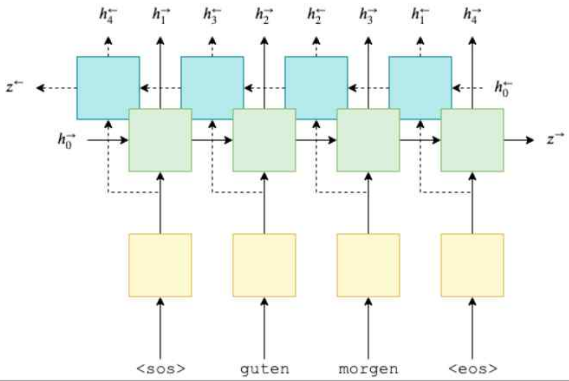
앞서 말한 논문들은 Machine Translation, 그중에서도 NMT를 다루고 있다. 이렇게 소개된 NMT의 구조는 주로 Encoder-Decoder방식을 사용하고 있는데, Encoder의 경우, 입력 문장을 고정 길이 벡터로 변환시키고, decoder는 해당 벡터를 이용해서 번역 결과를 생성해낸다. 해당 모델을 입력으로 제공된 Input sentence와 결과로 생성된 output sentence 사이의 probability를 최대화하는 방식으로 학습된다.

이러한 encoder-decoder 구조는 bottleneck 문제를 발생시킬 수 있다. Bottleneck 문제는 encoder에서 전체 문장을 하나의 고정된 길이의 벡터로 생성할 때 발생한다. 이러한 문제는 문장이 길어질수록 심각하게 나타나고, encoder-decoder 모델의 성능 또한 문장의 길이가 길어질수록 떨어지게 된다. 문장 전체의 정보를 짧은 고정 길이의 벡터로 모두 나타내기 어렵기 때문이다.

이 논문에서는 encoder-decoder 모델에서 새로운 구조를 추가해서 성능을 향상할 수 있는 방법을 제안한다. 해당 모델은 decoder에서 하나의 결과를 만들어낼 때마다, 입력 문장을 순차적으로 탐색해서 현재 생성하려는 부분과 가장 관련있는 영역을 적용시킨다. 최종적으로 encoder에서 생성한 context word 중 관련성이 크다고 판단되는 영역들과, decoder에서 이미 생성한 결과를 기반으로 다음 단어를 결과로 생성해낸다.

이러한 방식의 가장 큰 장점 중 하나는 입력 문장을 고정된 길이의 벡터로 표현하지 않아도 된다는 것이다. Decoder에서 연산을 진행하면서 encoder에서 생성한 context word를 계속해서 참조하기 때문에, 전체 문장의 정보를 하나의 벡터에 담으려고 하지 않아도 되고, 문장의 길이가 길어지더라도 성능을 유지할 수 있다.

align and translate (Attention)

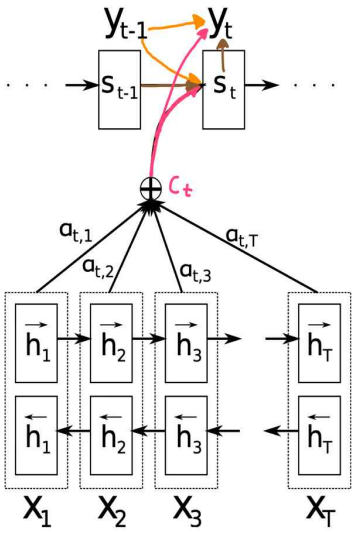


We now have:

$$h_i^{\rightarrow} = \text{EncoderGRU}^{\rightarrow}(e(x_i^{\rightarrow}), h_{i-1}^{\rightarrow})$$
$$h_i^{\leftarrow} = \text{EncoderGRU}^{\leftarrow}(e(x_i^{\leftarrow}), h_{i-1}^{\leftarrow})$$

기존 seq2seq model에서는 encoder는 1개의 context vector를 생성해내기 위해 존재했다. 하지만 본 model은 위에서 언급했듯이 하나의 context vector가 아닌 각 시점에 대한 context vector를 각각 생성한다. 이 때 사용하는 hjhj를 구하는 것이 encoder의 역할이다. hjhj는 위에서 언급했듯이 jj번째 input token에 대해 attention한 vector이다. 이 때 attention한다는 의미는 jj번째 input token에 대해 당연히 가장 높은 가중치를 주고, jj번째에서 멀어질수록 낮은 가중치를 주는 것이다. 이는 j−1j−1, j−2j−2 …의 역방향, j+1j+1, j+2j+2 …의 순방향, 즉 양방향에 대해 모두 적용되어야 한다. 이를 위해 사용한 것이 Bidirectional RNN이다. 순방향에 대한 hjhj를 생성하고, input sentence의 끝에 도달하면 다시 역방향에 대한 hJhJ를 생성한다. 그 후 순방향에 대한 hj→hj→와 역방향에 대한 hj←hj←를 함께 반영해 최종 hjhj를 만들어낸다.

Decoder



i번째 word로 y_i가 등장할 조건부 확률 : $p(y_i \mid y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$,

$s_i = f(s_{i-1}, y_{i-1}, c_i)$.

i시점의 hidden state — i-1시점의 hidden state, 직전 output word, i시점의 context vector

ii번째 token으로 yiyi가 등장할 조건부 확률에 대한 수식이다. y1∼yi−1y1∼yi−1 (yiyi 이전의 output sentence)와 XX(input sentence 전체)에 대해 다음 token으로 yiyi가 생성될 조건부 확률이다. 이는 gg 함수에 yi−1yi−1, SiSi, CiCi를 인자로 넣어 생성된 값이다. yi−1yi−1은 직전 시점 i−1i−1에서 생성한 output token이고, SiSi는 현재 시점 iiㅇ서의 RNN hidden state, CiCi는 현재 시점 i에 생성된 context vector이다. 직관적으로 해석해보면 이전 단어 yi−1yi−1 이후에 나올 단어 yiyi를 예측하는 것인데, 이 때 이전 output sentence의 상태 정보를 모두 포함하고 있는 SiSi를 입력으로 받음으로써 output sentence의 문맥을 반영하고, input sentence에 대한 context vector CiCi를 통해 input sentence의 문맥을 반영한다. 이전 seq2seq model에서는 context vector가 input sentence 전체에 대한 vector였는데, 이번 attention seq2seq model에서는 특정 시점 i에 대한 context vector CiCi가 주어진다. 즉, context vector가 input sentence 전체에 대한 하나의 vector가 아니라 각 시점 i에 대해 CiCi가 각각 정의된다는 것이다. 아래에서는 CiCi에 대해 좀 더 자세하게 살펴본다.

$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

CiCi는 aijaij와 hjhj에 대해 jj부터 TxTx까지 더한 vector이다. jj부터 TxTx까지의 의미는 input sentence의 처음부터 끝까지 각 input token에 대해 jj로 순회한다는 의미이다. 즉, jj는 input sentence에서의 token index이다. 반대로 ii는 output sentence에서 현재 시점 i이다. 정리하자면 jj는 input에 대한 index, ii는 output에 대한 index이다. 우리는 output의 ii 시점에서 생성되는 context vector CiCi에 대한 수식을 살펴보는 것이다. hjhj는 input sentence 전체의 context를 포함하지만 동시에 특히 jj번째 token과 그 주변에 대해 더 attention(집중)을 한 vector이다.aijaij는 iiㅂ째 output token이 jj번째 input token과 align될 확률값을 뜻한다. aijaij에 대해 더 살펴보자.

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)},$$

e_{ij}의 softmax

$$e_{ij} = a(s_{i-1}, h_j)$$

output의 i와 input의 j가 얼마나 서로 match하는가 / output에 i시점의 hidden state / input의 j에 focus

eijeij는 iі번째에 들어올 output token과 jj번째 input token이 얼마나 서로 잘 match되는지에 대한 값이다. output의 문맥을 반영하기 위해 Si−1si−1를 input으로 받고, jj번째 input token에 대한 attention을 주기 위해 hjhj를 input으로 받는다. 이렇게 완성된 eijeij를 softmax한 aijaij는 iі번째 들어올 output token과 jj번째 input token이 align될 확률값을 뜻한다.

다시Cici의 의미로 되돌아와보면, Cici는 iі 시점에서 모든 input sentence token jj에 대해 aijaij와 hjhj를 곱한 vector에 대한 합이다. aijaij는 현재 시점 aiai에서 생성될 output token이 jj번째 input token과 align될 확률값이며, hjhj는 input sentence에 대한 context이되, jj번째 input token에 특히 attention한 vector이다. 결론적으로 Cici는 현재 시점 iі에서 input sentence의 어느 token에 더 attention을 해야 하는지에 대한 context라고 직관적으로 해석 가능하다.

위의 model이 기존 seq2seq model에 비해 가지는 이점은 encoder가 고정된 size context vector 1개에 모든 input sentence token에 대한 attention을 담을 필요가 없다는 것이다. 왜냐하면 decoder model에서 단지 context vector 1개만 사용해 output sentence를 생성해내는 것이 아니라, hjhj와 같은 input sentence token에 대한 attention data를 사용하기 때문이다.