Neural Machine Translation by Jointly Learning to Align and Translate

해당 논문의 경우, seq2seq에 attention을 도입, NMT(Neural Machine translation, 신경망 기계 번역) task에 적용하여 성능을 발전시킨 논문이다.

Background

이 논문은 2015년도에 ICLR에 발표되었는데, 지금이야 NMT가 MT(Machine translation, 기계번역) task를 지배하고 있지만, 그 당시에는 이제 막 NMT가 발전하고 있던 시기였다.

NMT는 RNN계열의 아키텍쳐를 기반으로 한 Encoder-Decoder 구조로 수행되는데, 이 Encoder-Decoder 구조에는 문제가 있었다.

바로, fixed-length vector가 가지는 정보의 한계이다. 이것이 무슨 말인지 알아보기 위해 기존 구조의 NMT 수행 과정을 알아보겠다.

이 전까지의 Encoder-Decoder구조는 다음과 같은 과정으로 수행되었다.

1. Encoder에 source text가 input으로 투입되고, 학습되어 Encoder의 hidden state에 source text의 information이 encode
2. 이후 Encoder의 hidden state를 Decoder에 넘겨줌으로써 source text의 information 전달
3. 전달받은 information을 바탕으로 Decoder 학습

이 때, 우리는 source text의 information이 오로지 Encoder의 hidden state에만 담겨 전달되는 것을

확인할 수 있다.

Encoder의 hidden state는 고정된 길이의 벡터, 다시 말해 fixed-length vector인데, 이러한 구조는 source text의 전체 정보를 fixed-length vector에 전부 압축해야하는 구조인것이다.

그런데, 만약 엄청 긴 문장이 계속해서 나온다고 해보자. 과연 fixed-length vector인 Encoder의 hidden state가 모든 정보를 다 담을 수 있을까? 이는 불가능에 가깝다.

실제로, 논문에서는 해당 구조가 input sentence length가 길어질수록 성능이 향상된다는 연구 결과를 인용하면서, 이에 대한 문제를 제기한다.

논문에서는 이러한 문제를 해결하기 위해 새로운 방법을 제시한다.

바로, input sentence를 sequence of vector로 encoding한 다음, Decoder에서 decoding할 때 해당 encoding된 vector의 subset을 adaptively하게 선택하게끔 하는 방식이다.

즉, Decoder가 decoding하는동안 input sentence(source sentence)를 탐색하여 정보를 가져오는 것이다.

어떻게 이런 구조가 가능한 것일까? 하나씩 차근차근 살펴보자.

Learning to Align and Translate

우선, 전체 구조를 나타내는 그림과 함께 모델의 구조를 살펴보도록 하겠다.

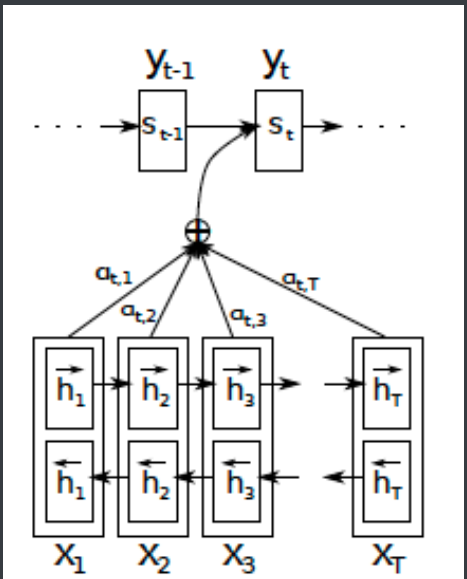

Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

논문에서는 Encoder에 bidirectional RNN을 사용하였고, 이로 인해 정방향과 역방향 모두 살펴볼 수 있어서 좋은 결과가 나올 수 있었다고 한다.

$$p(y_i|y_1 \ldots y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

위의 수식은 논문에서 제시하는 모델의 Decoder의 output을 나타낸다.

$p(y_i|y_1 \ldots y_{i-1}, \mathbf{x})$ 는, x와 $y_i|y_1 \ldots y_{i-1}$ 가 주어졌을 때,

y_i이 주어질 확률을 나타내는데, 이는 NMT에서 Decoding할 때(target language로 번역할 때) 이전까지의 정보를 바탕으로 현재 timestep i의 단어를 출력하는 것을 의미하며, 이를 논문에서 제시하는 모델에 맞게 변화시키면 $g(y_{i-1}, s_i, c_i)$가 된다.

g는 nonlinear를 나타내며, y_{i-1}는 이전 timestep의 output, s_i는 이전 timestep의 hidden state, c_i는 context vector를 의미하며 context vector의 경우 아래에서 더 자세히 설명하도록 하겠다.

정리하자면, 논문에서 제시하는 모델은 이전 timestep의 output, 이전 timestep의 hidden state, context vector를 바탕으로 output(번역할 단어)을 출력한다는 것이다.

논문에서는 기존 Encoder-Decoder 구조와 다르게, **각 단어 y_i(output)에 대한 확률은 각 대상 단어 y_i에 대한 context vector c_i에 따라 영향을 받는다는 것을 강조한다.**

그렇다면, 이 context vector c_i는 어떻게 구하는 것일까?

우선, 논문에서는 context vector c_i는 input sentence(source sentence)에 대한 Encoder map인 *annotation*에 의존한다고 말한다.

annotation이란, input sentence(source sentence)의 정보를 담고 있는 Encoder의 각 timestep별 hidden state이며, 각 timestep j의 annotation h_j는 j번째 단어를 둘러싼 부분에 초점을 맞춘 전체 input sentence(source sentence)에 대한 정보를 담고 있다.

context vector c_i는 이러한 annotation h_j의 weighted sum(가중 평균)으로 구해지는데, 과정은 다음과 같다.

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

여기서, α_{ij}는 annotation h_j의 weight(가중치)이며, 이는 다음과 같이 구한다.

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

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

annotation h_j의 weight α_{ij}는 e_ij에 softmax를 취해 얻을 수 있다.

e_ij는 alignment model이며, 이는 i번째 Decoder의 output과 j번째 Encoder의 input이 match가 잘 되는지의 점수이다.

이 점수는 Decoder의 $i-1$번째 timestep의 hidden state s_{i-1}와 j번째 annotation(Encoder의 j번째 hidden state) h_j를 alignment model α에 통과시켜 얻을 수 있다.

alignment model α에 통과시키는 과정은 논문의 Appendix에서 찾아볼 수 있는데, 다음과 같다.

$$\alpha(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$
$$\text{where } W_a \in \mathbb{R}^{n \times n}, U_a \in \mathbb{R}^{n \times 2n} \text{ and } v_a \in \mathbb{R}^n \text{ are the weight matrices}$$

논문에서는 이 alignment model α는 feed-forward network로, 기존의 방법론들과 다르게 soft alignment를 직접 계산하고, cost function의 gradient를 이용한 backpropagate로 optimize할 수 있다고 언급한다.

잠깐 다른 길로 새보자면, 수식을 보면서 생각나는 TMI인데, 아까 잠깐 언급한대로, Encoder에 Bidirectional RNN을 사용하여 annotation h_j에 곱해지는 weight matrix의 shape이 $n \times 2n$인것이 눈에 띈다.

또한, Encoder 부분은 autoregressive하지 않기에, $U_a h_j$ 부분은 사전에 한번에 계산하여 computational cost를 줄일 수 있다.

어찌되었든, Decoder의 $i-1$번째 timestep의 hidden state s_{i-1}와 annotation h_j에 각각 weight matrix를 곱하고, 여기에 탄젠트-하이퍼볼릭 비선형 함수를 적용시킨 이후, 마지막으로 weight matrix v_a를 곱해주면서

i번째 Decoder의 output과 j번째 Encoder의 input이 match가 잘 되는지의 점수, e_ij를 구하는 여정이 끝나게 된다.

여기서 이 alignment model α에 대해 짚고 넘어갈 점이 있다.

1. 3개의 neural network layer(weight matrices)를 사용하였다
2. timestep t에 대한 output을 구할 때, timestep $t-1$의 hidden state를 이용하여 context vector를 구하였다.

이 두 항목은 추후 다른 논문에서 보완되며, 해당 내용은 다른 게시물에서 다루도록 하겠다.

이후로는, Encoder의 각 j번째 timestep마다 이 점수를 구한 뒤, softmax를 취해주어 는 annotation h_j의 weight(가중치) α_{ij}를 구하고, 이를 annotation h_j에 곱하면 context vector c_i를 구하는 길고 긴 여정이 끝나게 된다.

다음의 과정을 시간 순서대로 정리해보자

1. Decoder의 $i-1$번째 timestep의 hidden state s_{i-1}와 annotation h_j에 각각 weight matrix를 곱함
2. 탄젠트-하이퍼볼릭 비선형 함수를 적용
3. weight matrix v_a를 곱함, i번째 Decoder의 output과 j번째 Encoder의 input이 match가 잘 되는지의 점수, e_ij를 구함
4. e_ij에 softmax를 취해 annotation h_j의 weight α_{ij}를 구함
5. annotation h_j의 weight(가중치) α_{ij}를 구하고, 이를 annotation h_j에 곱하여 context vector c_i 구함

이렇게 구해진 e_ij와 α_{ij}는, Encoder의 각 annotation h_j이 현재 timestep i의 hidden state와 y를 결정하는 이전 timestep $i-1$의 hidden state에 얼마나 중요한지 나타낸다.

논문에서는 이러한 attention mechanism을 통해, 논문의 서두에서 언급한 fixed-length vector인 Encoder의 hidden state가 모든 정보를 다 담을 수 없다는 문제를 해결할 수 있다고 주장한다.

이후, 이렇게 구해진 context vector는 다음과 같은 과정을 거쳐 Decoder의 i번째 hidden state인 s_i로 탈바꿈하게 된다.

$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i,$$

where

$$\tilde{s}_i = \tanh\left(WEy_{i-1} + U\left[r_i \circ s_{i-1}\right] + Cc_i\right)$$
$$z_i = \sigma\left(W_z Ey_{i-1} + U_z s_{i-1} + C_z c_i\right)$$
$$r_i = \sigma\left(W_r Ey_{i-1} + U_r s_{i-1} + C_r c_i\right)$$

E is the word embedding matrix for the target language. $W, W_z, W_r \in \mathbb{R}^{n \times m}$, $U, U_z, U_r \in \mathbb{R}^{n \times n}$, and $C, C_z, C_r \in \mathbb{R}^{n \times 2n}$ are weights. Again, m and n are the word embedding dimensionality and the number of hidden units, respectively. The initial hidden state s_0 is computed by $s_0 = \tanh\left(W_s \overleftarrow{h}_1\right)$, where $W_s \in \mathbb{R}^{n \times n}$.

이 또한 Appendix에서 찾아볼 수 있다.

여기서도 Bidirectional RNN으로 인해 Context vector에는 $n \times 2n$의 weight matrix를 곱해주는 것이 눈에 띈다.

Experiment and Result

Model	All	No UNK°
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50*	28.45	36.15
Moses	33.30	35.63

Table 1: BLEU scores of the trained models computed on the test set. The second and third columns show respectively the scores on all the sentences and, on the sentences without any unknown word in themselves and in the reference translations. Note that RNNsearch-50* was trained much longer until the performance on the development set stopped improving. (○) We disallowed the models to generate [UNK] tokens when only the sentences having no unknown words were evaluated (last column).

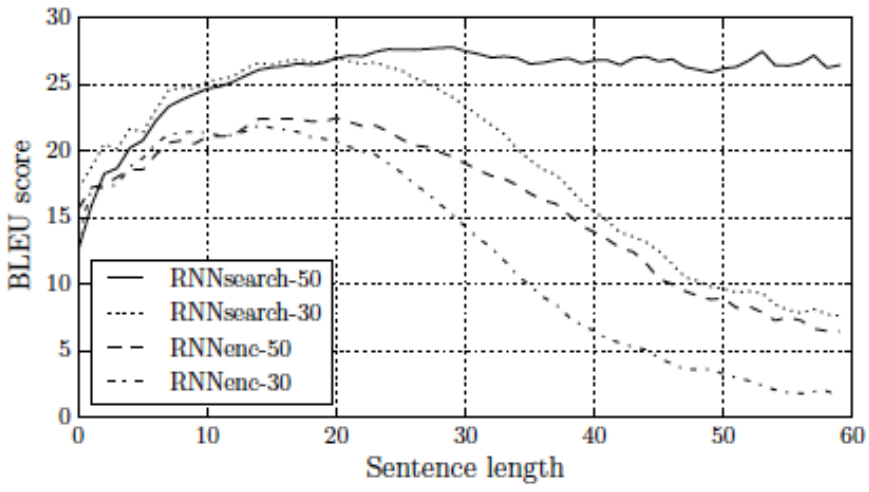

Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

위 자료들에서 RNNsearch가 이제까지 살펴본 모델이다. 기존 구조에 비해서 좋은 성능을 보임을 확인할 수 있다.

특히, 아래의 그래프를 통해 문장 길이가 늘어날수록 기존 구조와는 압도적인 성능 차이가 있음을 확인할 수 있다.

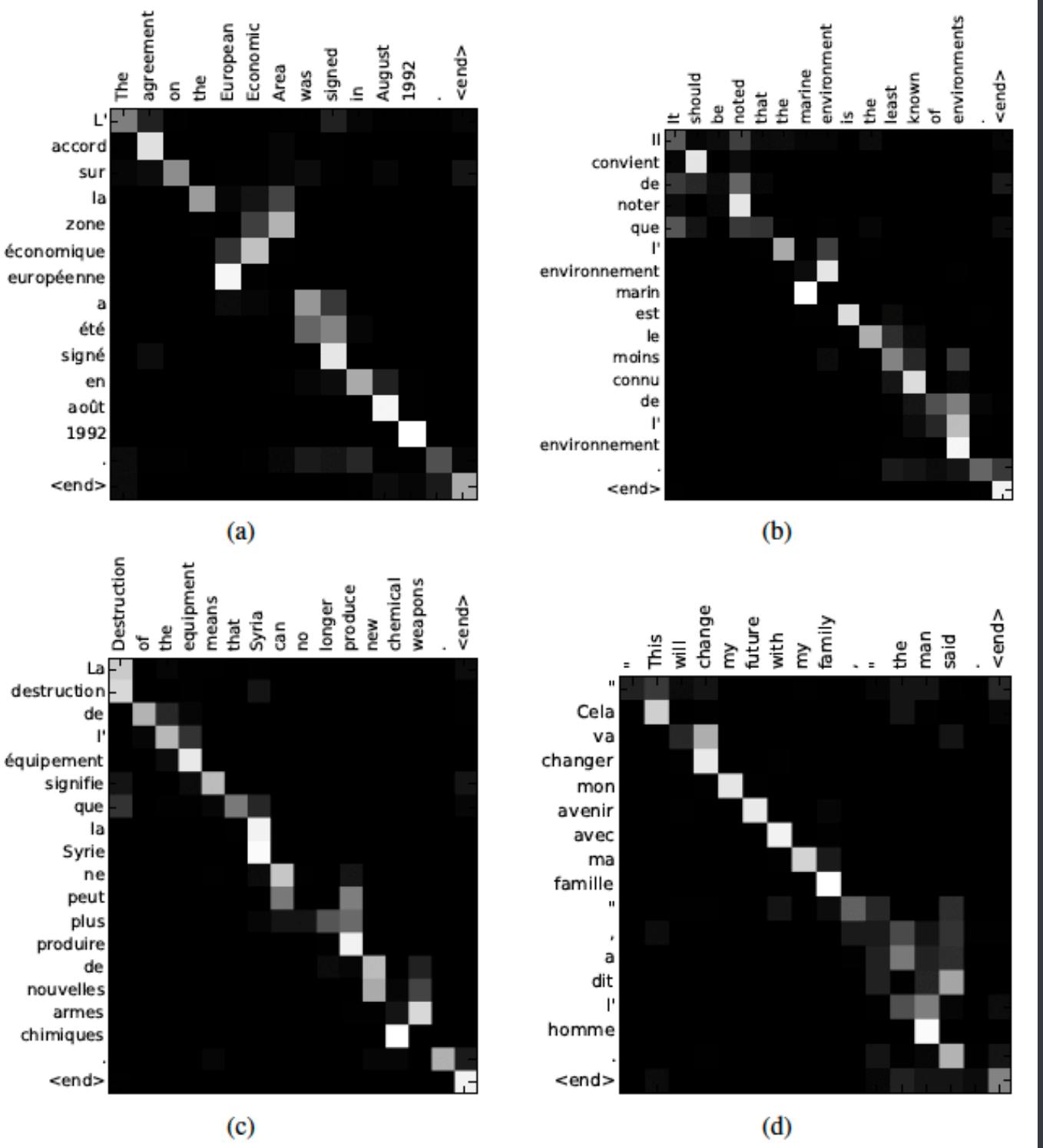

Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word (see Eq. (6)), in grayscale (0: black, 1: white). (a) an arbitrary sentence. (b–d) three randomly selected samples among the sentences without any unknown words and of length between 10 and 20 words from the test set.

또한, 위 자료는 target word를 생성할 때, input(source) sequence에서 어떤 단어를 중요하게 계산했는지를 시각화한 자료이다.

대부분 한 위치마다 한 단어가 align되어있는 것을 확인할 수 있다. 이로 인해 해당 논문이 제안한 alignment model이 효과적으로 동작함을 확인할 수 있었다.