Word-Piece Model

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

AI Robotics KR NLP Study
정지용

Word-Piece Model?

- Out of Vocabulary(OOV) Problem
  - Neural Machine Translation 은 고정된 크기의 단어 사전 사용
  - 사전에 없는 단어가 등장하는 경우, 이 단어를 처리하는 방안 필요

- OOV words 를 translation하기 위한 두가지 접근 방안
  1) simply copy rare words
  2) Sub-word units
     1) Character Model
     2) Mixed Word & Character Model
     3) **Wordpiece Model**

- Balance between the flexibility of characters and efficiency of words

cf) Mixed Word & Character Model

- 고정된 크기의 단어 사전에 포함된 단어는 단어로 사용하고

  그외의 OOV 단어는 character 로 변환하되 prefix 태그를 붙여서 사용
  - 시작문자 : <B>
  - 중간문자 : <M>
  - 끝 문자 : <E>

ex) Miki

<B>M <M>i <M>k <E>i

- Source & target sentence 둘다 적용
- 후처리 단계에서 prefix를 제거하여 원래 단어로 tokenization

Generate Word-Piece Model

- language-model likelihood 를 최대화하는 방법으로 학습
- Byte Pair Encoding 알고리즘(Sennirch at el, 2015) 과 유사
- 글자단위에서 점차적으로 단어 집합을 만들어내는 Bottom-up 방식

- Step
  1) 초기 단어 집합을 Unicode character 집합으로 변환하여 inventory 구성
  2) inventory에 있는 유니그램을 쌍을 구성
  3) 초기 단어 집합에 빈도수가 가장 높은 쌍을 하나의 유니그램으로 변환하여 inventory 재구성
  4) 미리 정한 단어 유닛의 갯수에 도달할때까지 2), 3) 과정 반복

Generate Word-Piece Model

- 예시 (참고 : https://wikidocs.net/22592)

```
# dictionary
l o w : 5,  l o w e r : 2,  n e w e s t : 6,  w i d e s t : 3
```

```
# vocabulary
l, o, w, e, r, n, w, s, t, i, d
```

```
# dictionary update!
l o w : 5,
l o w e r : 2,
n e w es t : 6,
w i d es t : 3
```

```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es
```

Generate Word-Piece Model

- 예시 (참고 : https://wikidocs.net/22592)

```
# dictionary update!
l o w : 5,
l o w e r : 2,
n e w est : 6,
w i d est : 3
```

```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es, est
```

```
# dictionary update!
lo w : 5,
lo w e r : 2,
n e w est : 6,
w i d est : 3
```

```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es, est, lo
```

Generate Word-Piece Model

- 예시 (참고 : https://wikidocs.net/22592)

```
# dictionary update!
low : 5,
low e r : 2,
newest : 6,
widest : 3
```

```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es, est, lo, low, ne, new, newest, wi, wid, widest
```

Apply Word-Piece Model

- 미리 학습된 Word-Piece Model (WPM) 을 토대로 단어 분리
- 예시)

Original) Jet makers feud over seat width with big orders at stake

WPM) _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

- 단어의 시작에는 _ 붙임

- 실험 결과 8k ~ 32k 정도로 단어를 분리했을 때, 학습 성능 및 정확도가 좋았음

Experiments

- 212K Source words 와 80k Target words 로 구성된 단어 사전으로 실험

Table 4: Single model results on WMT En→Fr (newstest2014)

Model	BLEU	CPU decoding time per sentence (s)
Word	37.90	0.2226
Character	38.01	1.0530
WPM-8K	38.27	0.1919
WPM-16K	37.60	0.1874
WPM-32K	38.95	0.2118
Mixed Word/Character	38.39	0.2774
PBMT [15]	37.0	
LSTM (6 layers) [31]	31.5	
LSTM (6 layers + PosUnk) [31]	33.1	
Deep-Att [45]	37.7	
Deep-Att + PosUnk [45]	39.2	

Table 5: Single model results on WMT En→De (newstest2014)

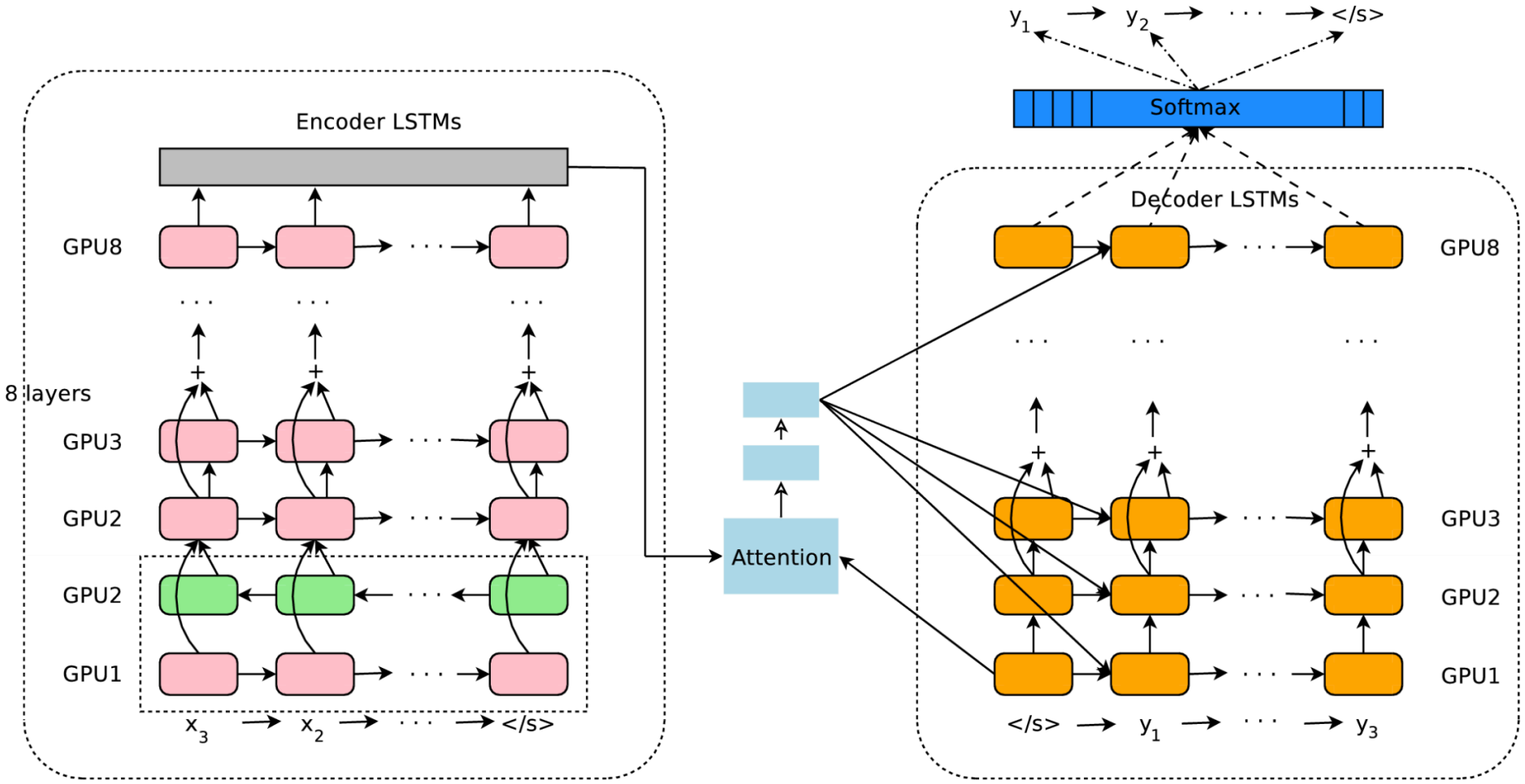
Model	BLEU	CPU decoding time per sentence (s)
Word	23.12	0.2972
Character (512 nodes)	22.62	0.8011
WPM-8K	23.50	0.2079
WPM-16K	24.36	0.1931
WPM-32K	24.61	0.1882
Mixed Word/Character	24.17	0.3268
PBMT [6]	20.7	
RNNSearch [37]	16.5	
RNNSearch-LV [37]	16.9	
RNNSearch-LV [37]	16.9	
Deep-Att [45]	20.6	

Google’s Neural Machine Translation System

- Neural Machine Translation 이 실제 서비스에서 사용되기 힘든 이유 해결
- 기존 NMT의 문제점
  - training 과 Inference 할때 많은 시간과 연산 리소스 필요
  - Rare word 에 대한 처리 문제
  - Input sentence 전체를 커버하지 않고 번역하는 문제

- GNMT 의 특징
  - 8개의 LSTM Encoder 와 8개의 LSTM Decoder 그리고 Attention 모듈
  - 학습속도를 올리기 위해, 모델 및 데이터 병렬 구조
  - Inference 속도를 올리기 위해, low-precision 연산 사용
  - Rare word 에 대한 처리를 word-piece model 로 해결
  - beam search 알고리즘을 통해 output sentence 가 input 의 모든 단어를 커버하도록 함

Model Architecture



Model Architecture

- X : Sequence of Source Sentence $X = x_1, x_2, x_3, ..., x_M$
- Y : Sequence of Target Sentence $Y = y_1, y_2, y_3, ..., y_N$

$$\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_M} = EncoderRNN(x_1, x_2, x_3, ..., x_M)$$

$$P(Y|X) = P(Y|\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ..., \mathbf{x_M})$$

$$= \prod_{i=1}^{N} P(y_i|y_0, y_1, y_2, ..., y_{i-1}; \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ..., \mathbf{x_M})$$

- $\mathbf{a}_i$: context vector

$$s_t = AttentionFunction(\mathbf{y}_{i-1}, \mathbf{x}_t) \quad \forall t, \quad 1 \leq t \leq M$$

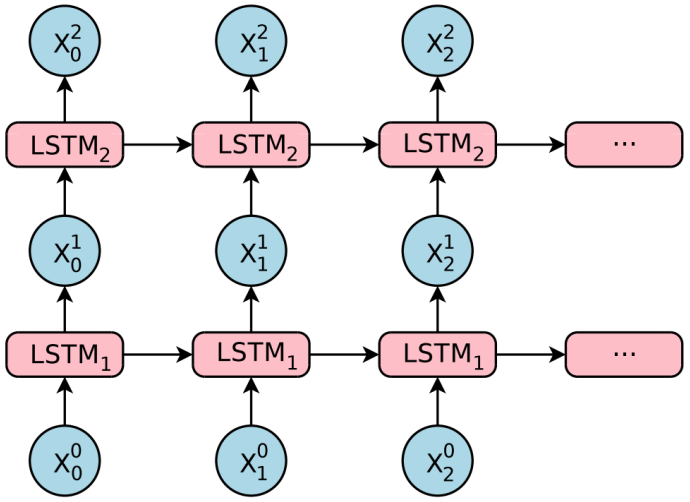
$$p_t = \exp(s_t) / \sum_{t=1}^{M} \exp(s_t) \qquad \forall t, \quad 1 \leq t \leq M$$

$$\mathbf{a}_i = \sum_{t=1}^{M} p_t . \mathbf{x}_t$$

Residual Connections

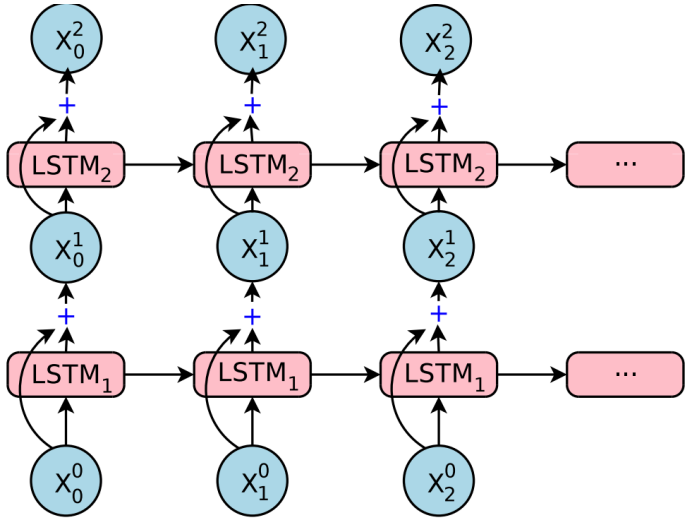
기존

$$\mathbf{c}_t^i, \mathbf{m}_t^i = \text{LSTM}_i(\mathbf{c}_{t-1}^i, \mathbf{m}_{t-1}^i, \mathbf{x}_t^{i-1}; \mathbf{W}^i)$$
$$\mathbf{x}_t^i = \mathbf{m}_t^i$$
$$\mathbf{c}_t^{i+1}, \mathbf{m}_t^{i+1} = \text{LSTM}_{i+1}(\mathbf{c}_{t-1}^{i+1}, \mathbf{m}_{t-1}^{i+1}, \mathbf{x}_t^i; \mathbf{W}^{i+1})$$



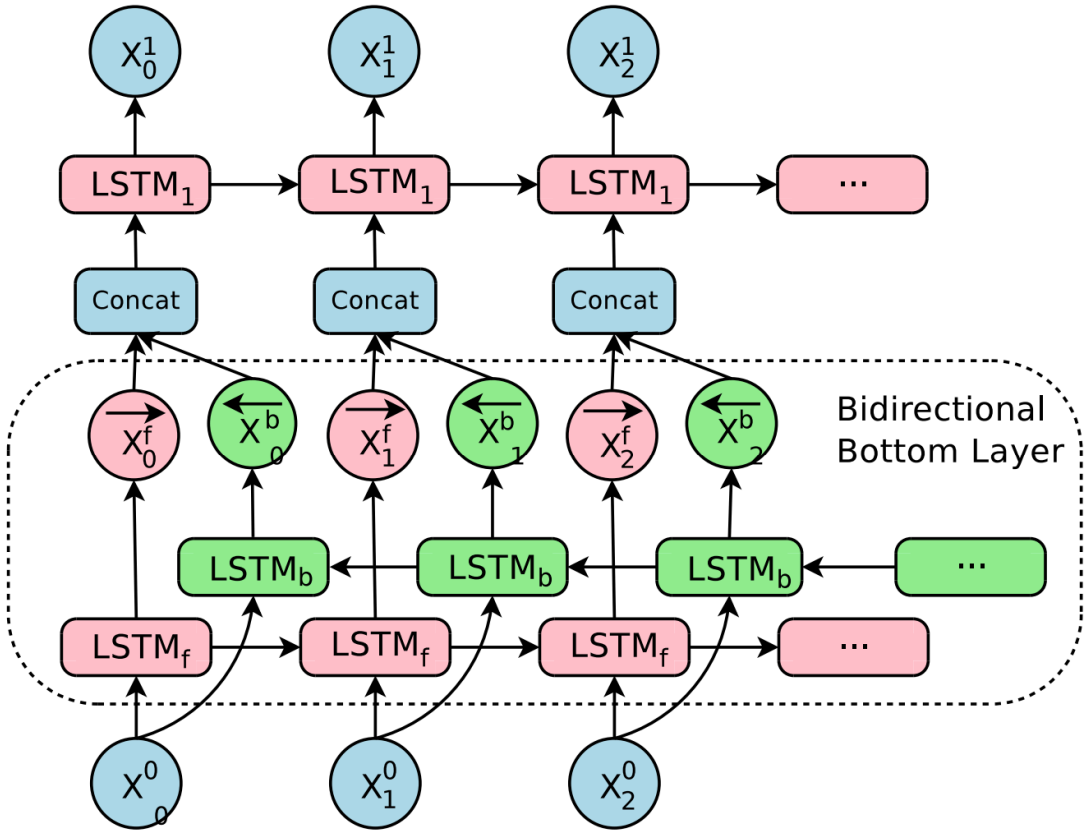
Residual Connections

$$\mathbf{c}_t^i, \mathbf{m}_t^i = \text{LSTM}_i(\mathbf{c}_{t-1}^i, \mathbf{m}_{t-1}^i, \mathbf{x}_t^{i-1}; \mathbf{W}^i)$$
$$\mathbf{x}_t^i = \mathbf{m}_t^i + \mathbf{x}_t^{i-1}$$
$$\mathbf{c}_t^{i+1}, \mathbf{m}_t^{i+1} = \text{LSTM}_{i+1}(\mathbf{c}_{t-1}^{i+1}, \mathbf{m}_{t-1}^{i+1}, \mathbf{x}_t^i; \mathbf{W}^{i+1})$$



Bi-directional Encoder for First Layer

- 충분히 context 를 고려하기 위해 bottom encoder layer 에만 적용



Model Parallelism

- Training 속도를 올리기 위해 모델 병렬화와 데이터 병렬화 모두 사용

- 데이터 병렬화
  - Downpour SGD 이용
  - 모델 파라미터를 모든 장비가 공유
  - Adam 과 SGD 결합된 알고리즘을 이용하여 비동기적으로 모델 파라미터 업데이트

- 모델 병렬화
  - 다중 GPU 장비를 활용하여 각 레이어마다 다른 GPU 할당
  - Bottom layer 말고는 모두 uni-directional 구조이기 때문에
    i번째 레이어가 다끝나기 전에 i+1 레이어 시작 가능

Training Criteria

- N 개의 input-output sequence pairs $\mathcal{D} \equiv \{(X^{(i)}, Y^{*(i)})\}_{i=1}^{N}$

$$\mathcal{O}_{\mathrm{ML}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P_\theta(Y^{*(i)} \mid X^{(i)})$$

- Objective function이 평가지표 (BLEU) 를 제대로 반영 못하는 문제
- Reward 도입 $r(Y, Y^{*(i)})$

$$\mathcal{O}_{\mathrm{RL}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \sum_{Y \in \mathcal{Y}} P_\theta(Y \mid X^{(i)})\ r(Y, Y^{*(i)})$$

$$\mathcal{O}_{\mathrm{Mixed}}(\boldsymbol{\theta}) = \alpha * \mathcal{O}_{\mathrm{ML}}(\boldsymbol{\theta}) + \mathcal{O}_{\mathrm{RL}}(\boldsymbol{\theta})$$

Quantizable Model and Quantized Inference

- inference 시 속도를 올리기 위해 사용
- 소수점 연산을 8bit 혹은 16bit 고정소수점 정수로 변환하여 연산
- quantization error 를 줄이기 위해 constraints 추가

$$
\begin{aligned}
\mathbf{c'}_t^i, \mathbf{m}_t^i &= \mathrm{LSTM}_i(\mathbf{c}_{t-1}^i, \mathbf{m}_{t-1}^i, \mathbf{x}_t^{i-1}; \mathbf{W}^i) \\
\mathbf{c}_t^i &= \max(-\delta, \min(\delta, \mathbf{c'}_t^i)) \\
\mathbf{x'}_t^i &= \mathbf{m}_t^i + \mathbf{x}_t^{i-1} \\
\mathbf{x}_t^i &= \max(-\delta, \min(\delta, \mathbf{x'}_t^i)) \\
\mathbf{c'}_t^{i+1}, \mathbf{m}_t^{i+1} &= \mathrm{LSTM}_{i+1}(\mathbf{c}_{t-1}^{i+1}, \mathbf{m}_{t-1}^{i+1}, \mathbf{x}_t^i; \mathbf{W}^{i+1}) \\
\mathbf{c}_t^{i+1} &= \max(-\delta, \min(\delta, \mathbf{c'}_t^{i+1}))
\end{aligned}
$$

Decoder

- decoding 과정에서 Beam search를 사용하여 score function을 최대화 하는 Y 탐색
- 두가지 정제 로직 사용
  - Converage penalty : input 데이터를 충분히 사용하였는지 판단
  - Length normalization : 길이가 다른 문장에 대한 정규화 (긴 문장이 손해보기 때문에)

$$s(Y,X) = \log(P(Y|X))/lp(Y) + cp(X;Y)$$

$$lp(Y) = \frac{(5+|Y|)^{\alpha}}{(5+1)^{\alpha}}$$

$$cp(X;Y) = \beta * \sum_{i=1}^{|X|} \log(\min(\sum_{j=1}^{|Y|} p_{i,j}, 1.0)),$$

감사합니다