Information from parts of words: Subword Models

Wongyu Kim (김원규)

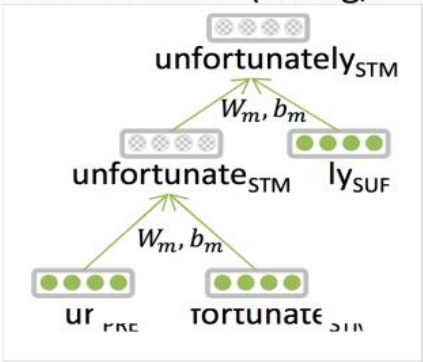
Internet Computing Laboratory
Department of Computer Science
Yonsei University

2020.09.17

Semantic Morphology

Morphology: Parts of words

- Traditionally, we have morphemes as smallest **semantic** unit
  - [[un [[fortun(e)]$_{ROOT}$ ate]$_{STEM}$]$_{STEM}$ ly]$_{WORD}$
- Deep learning: Morphology little studied; one attempt with recursive neural networks is (Luong, Socher, & Manning 2013)



A possible way of dealing with a larger vocabulary – most unseen words are new morphological forms (or numbers)

5

✓ 한 단어 내에서도 여러 semantic한 의미를 가지도록 나눌 수 있고 그 단위에 따라 임베딩 해야 할 과제가 남아 있다.(이전까지는 word 단위로 주로 임베딩 하였다.)

✓ 즉 morpheme 단위(형태소)로 하는 등, subword 단위로 임베딩 모델을 구성하는 것이 목표

Word vs Char Level Embedding

Word-Level Embedding

- ✓ 보통 단어를 one-hot 벡터로 바꾼 뒤 모델에 입력한다.
- ✓ 따라서 vocab 의존적이며 vocab에 없는 단어를 one-hot 벡터로 바꾸기 위해서는 <UNK> 단어로 바뀐다. = Out Of Vocab 문제
- ✓ 단어의 유사성은 파악할 지라도 contextual meaning은 파악하지 않는다.(ELMo 전에)
- ✓ 또한 언어 종류마다 단어 특성이 달라 word 단위로 나누기 애매할 때가 있다.

Char-Level Embedding

- ✓ OOV 문제가 생기지 않는다. (문자 단위라서 vocab에 의존적이지 않기 때문이다. 각 문자를 매핑하는 dict는 필요할 지라도 word-level 만큼 많은 단어가 필요한 건 아니다.(하지만 한글의 경우 글자 조합이 많기 때문에 애매할 수도))
- ✓ 다양한 언어에 걸쳐 유사한 스펠링들은 유사한 임베딩을 가진다.
- ✓ Char 단위로 하기 때문에 다양한 언어의 특성을 어느정도 일반화 시킬 수 있다.
- ✓ 하지만 모델에 입력 시 문자 하나 당 임베딩 한 개가 필요하니 그만큼 많은 벡터들이 필요하고 학습 & 테스트 시에 느릴 것이다.
- ✓ 또한 char 단위로 임베딩하면 그냥 그 문자만 보기 때문에 word 내에서 semantic이 떨어진다.

Word vs Char Level Embedding

English-Czech WMT 2015 Example

source	Her *11-year-old* daughter , *Shani Bart* , said it felt a little bit *weird*
human	Její **jedenáctiletá** dcera **Shani Bartová** prozradila , že je to trochu **zvláštní**
char	Její **jedenáctiletá** dcera , **Shani Bartová** , říkala , že cítí trochu ***divně***
word	Její <unk> dcera <unk> <unk> řekla , že je to trochu divné
	Její **11-year-old** dcera **Shani ,** řekla , že je to trochu ***divné***

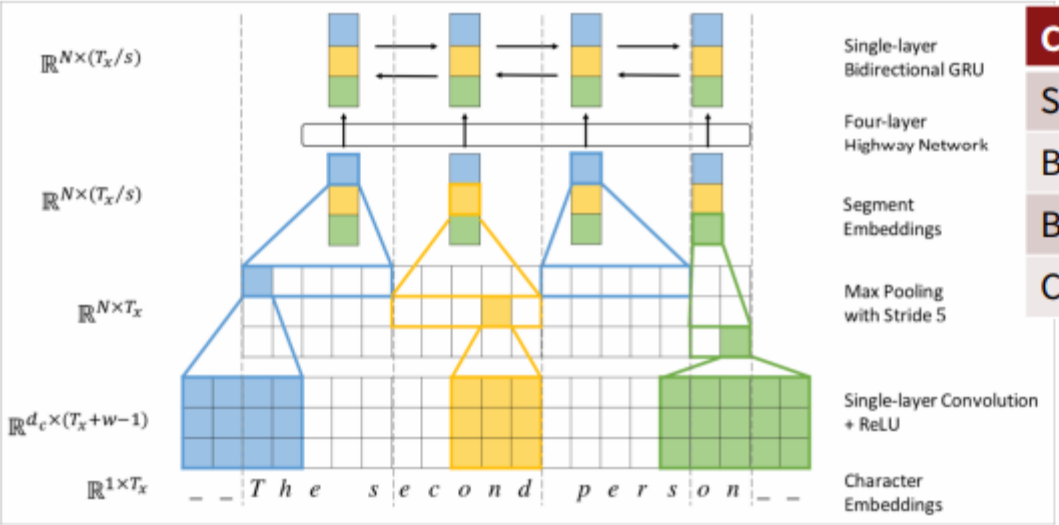
System	BLEU
Word-level model (single; large vocab; UNK replace)	15.7
Character-level model (single; 600-step backprop)	15.9

Word vs Char Level Embedding

Fully Character-Level Neural Machine Translation without Explicit Segmentation

Jason Lee, Kyunghyun Cho, Thomas Hoffmann. 2017.

Encoder as below; decoder is a char-level GRU



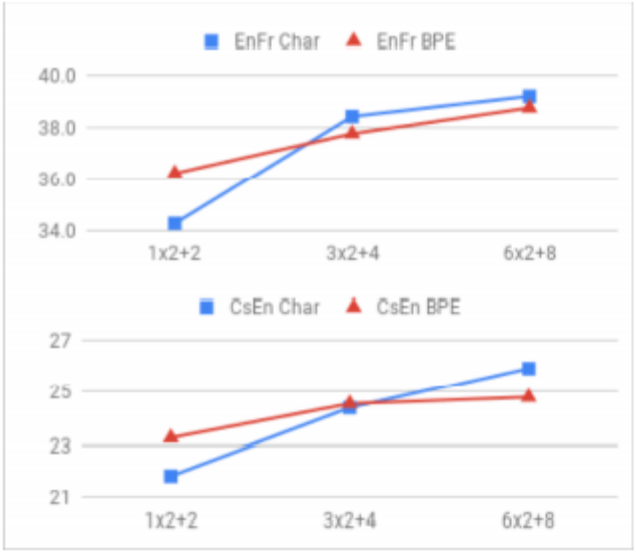
CS-En	WMT 15	Test
Source	Target	BLEU
Bpe	Bpe	20.3
Bpe	Char	22.4
Char	Char	22.5

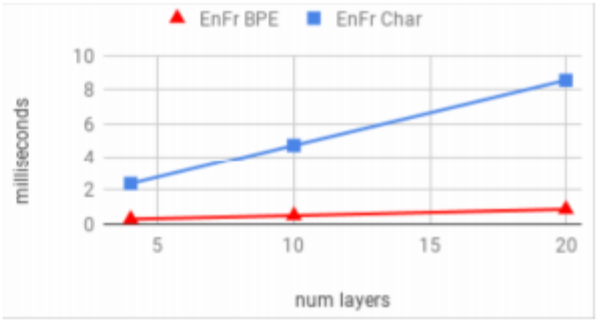
15

Word vs Char Level Embedding

Stronger character results with depth in LSTM seq2seq model

Revisiting Character-Based Neural Machine Translation with Capacity and Compression. 2018. Cherry, Foster, Bapna, Firat, Macherey, Google AI





Sub-word models: two trends

그렇다면 현재 임베딩을 하는 방법은 어떤 것이 trend 일까? (이전까지는 GloVe, Word2Vec, Char-level embedding 많이 사용했었다.)

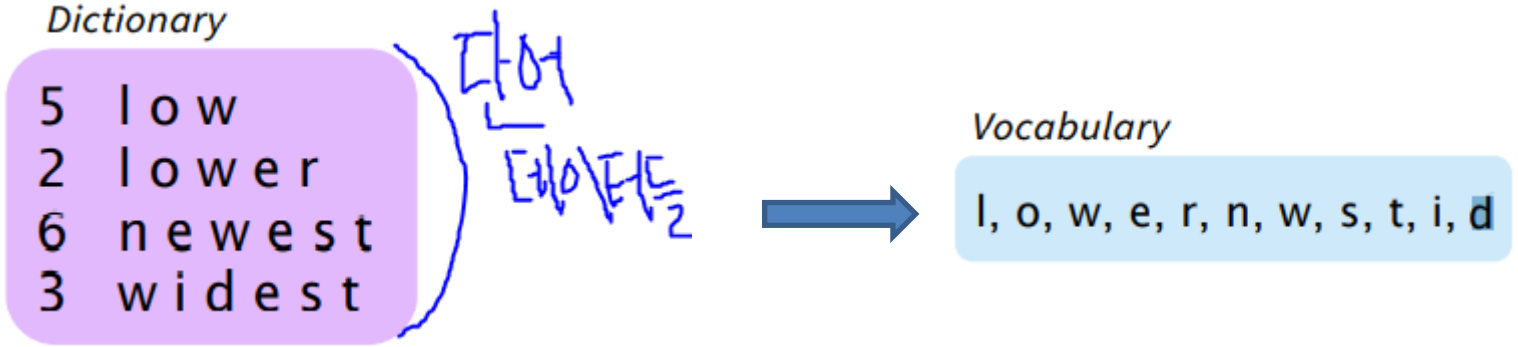
1. "Word Pieces" (Same Architecture as for word-level model, but smaller units)
2. Hybrid Architecture (main = words, support = char)

Byte Pair Encoding

- ✓ “Word Pieces”의 기반이 된다.
- ✓ Word-Level 과 Char-Level의 중간 정도라고 생각하면 된다. (= byte-level, 또한 그에 맞는 새로운 vocab도 구성하게 된다.)
- ✓ Word를 subwords로 분리하여 임베딩하는 것이 기본이다.
- ✓ OOV를 완전히 해결은 못할지라도 매우 많이 완화할 수 있다.

Byte Pair Encoding Algorithm

✓ 처음에는 우리가 알고있는 단어들 모두에 대해 char 단위로 Vocab을 구성한다.

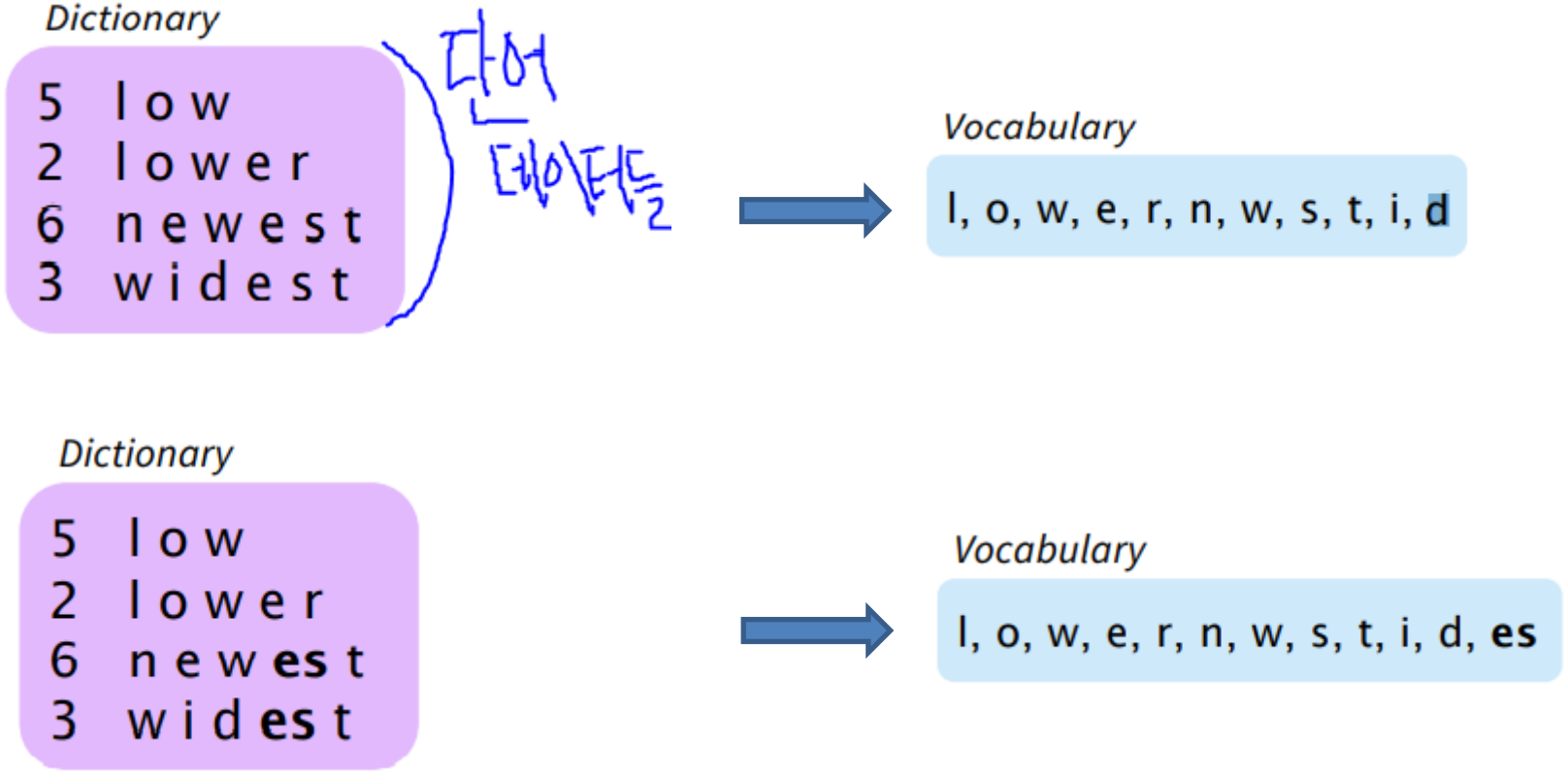


✓ low 라는 단어가 5개, lower라는 단어가 2개 … Widest라는 단어가 3개 있다는 뜻

✓ 그리고 차차 Vocab을 늘려가는 것 뿐이다. (Vocab을 Byte 단위로 새롭게 늘려가는게 핵심)

✓ 그렇다면 어떻게 vocab을 늘려갈까?

Byte Pair Encoding Algorithm



- ✓ Bi-gram으로 따져서 어떤 쌍이 제일 많은 지 찾으면 된다.
- ✓ 위 그림에서 (e, s)의 쌍은 9번 나오며 가장 많다. 그 뒤로는 (l, o) 쌍, (o, w) 쌍이 뒤따른다.
- ✓ 따라서 es를 한 문자로 여겨 vocab에 추가한다.

Byte Pair Encoding Algorithm

Dictionary

5 l o w
2 l o w e r
6 n e w **es** t
3 w i d **es** t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, **es**

Dictionary

5 l o w
2 l o w e r
6 n e w **est**
3 w i d **est**

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, **est**

- ✓ 그 다음은 es를 한 문자라 취급하고 다시 똑같은 작업을 하면 된다.
- ✓ 이번에는 (es, t) 쌍이 9번으로 제일 많고 이 est 문자를 vocab에 추가한다.
- ✓ 이런 식으로 사용자가 지정한 횟수만큼 진행하면 된다.

Byte Pair Encoding Algorithm

Dictionary

```
5  l o w
2  l o w e r
6  n e w e s t
3  w i d e s t
```

단어 데이터들

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

- ✓ 우리가 기존에 알고 있던 단어는 위에처럼 [low, lower, newest, widest] 이다.
- ✓ 하지만 "lowest" 라는 단어를 임베딩 해야할 때
  기존 Word-Level 에서는 Vocab에 없으니 <UNK>로 여길 것이다.
  기존 Char-Level 에서는 l,o,w,e,s,t 따로 임베딩해서 너무 많은 벡터가 있을 것이다
  하지만 BPE 에서는 "low"와 "est"를 한 문자(Byte)로 취급해 lowest를 low와 est로 따로 임베딩 할 것이다.
- ✓ WPM(Word Piece Model), SPM(Sentence Piece Model)의 기반이 된다.
- ✓ BERT와 GPT에서 BPE를 기반으로 임베딩한다. (BERT는 WPM을, GPT는 BPE를)

Word Piece Model

- ✓ BPE랑 비슷한데 어떤 Byte를 우선 순위로 Vocab에 추가할지에 대한 알고리즘이 바뀌었다.
- ✓ 기존에는 빈도수로 따졌다면 여기서는 코퍼스의 likelihood를 가장 높이는 그 쌍을 병합한다.
  (자세한 알고리즘은 생략)

WPM을 수행하기 이전의 문장: Jet makers feud over seat width with big orders at stake
WPM을 수행한 결과(wordpieces): _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

- ✓ 일단 여기서 모든 단어 앞에는 _ 를 붙이고 시작 (문장 복원을 위한 flag 장치)
- ✓ 그 상태로 WPM을 하면 아래 문장 처럼 나뉜다.
  (ex) vocab = { _J: 1, et: 2, _makers: 3, _fe: 4, ud: 5, _over: 6 … }
- ✓ 그런데 _J과 et 사이의 띄어쓰기가 _Jet과 _makers의 띄어쓰기
  (즉, word 내의 띄어쓰기 vs words 간의 띄어쓰기) 인지 구별 어떻게?
  => _를 이용하면 어떤 띄어쓰기 인지 알 수 있다.

Unigram

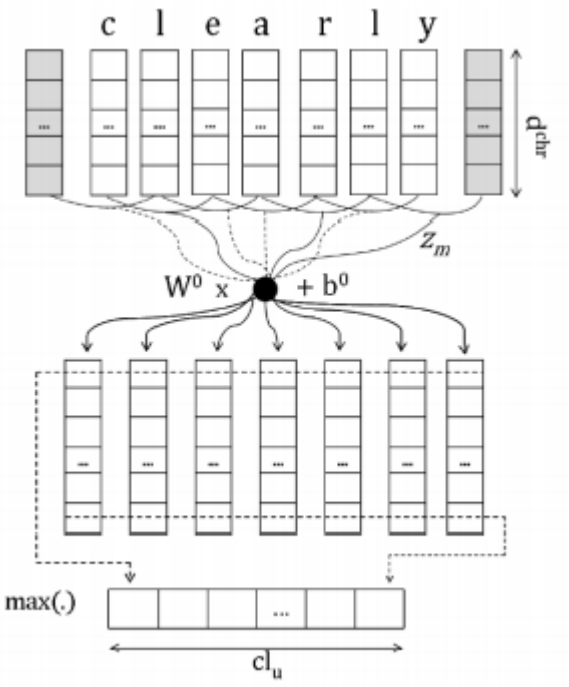
- ✓ BPE or WPM으로(pre-tokenizer로) Vocab을 구성했다 치자.
- ✓ 거기서 필요 없다고 생각되는 unit(byte)를 제거하는 과정이다.
- ✓ 즉 주어진 코퍼스와 현재 Vocab에 대한 Loss를 계산하는 것이다.
- ✓ 이때 Vocab에서 subword가 제거되었을 때 Loss가 조금 올라간다면 그 subword는 제거하면 안된다.
- ✓ Vocab의 10%~20%를 제거하게 된다.
- ✓ 기본 캐릭터들은 제거되지 않고 반드시 유지된다.(a,b,c,d ...)

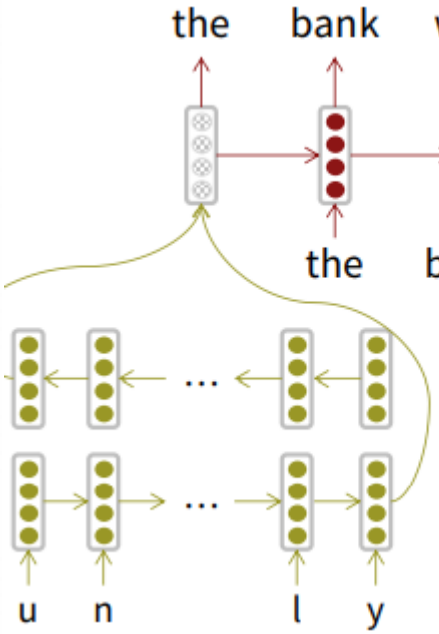
Sentence Piece Model

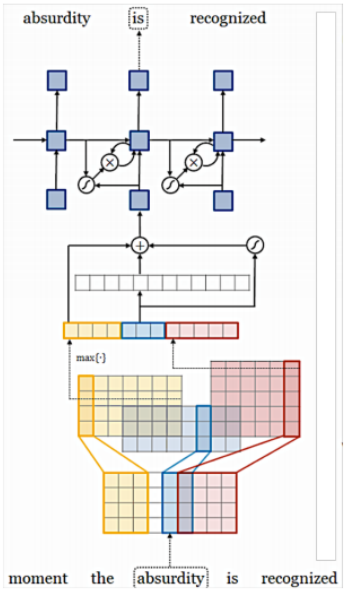
- ✓ XLNet, ALBERT, Marian NMT 에서 사용하는 것이다.(WPM 보다 더 최근)
- ✓ BPE, WPM은 그 자체가 pre-tokenize를 하는 것이고, Unigram은 pre-tokenize를 필요로 한다.
- ✓ 하지만 WPM에서 봤듯이 모든 단어 앞에 _를 붙이게 되는데 어떤 언어는 띄어쓰기로 단어를 구분하지 않는다.
- ✓ 따라서 모든 언어에 대해 바로 RAW 한 코퍼스를 tokenize 할 수 있게 한다.
- ✓ 원리는 BPE + Unigram을 적용한다.
- ✓ 따라서 단어를 구분할 수 있게 special charter(‘_’) 을 이용한다. (로직은 모르겠지만 이렇게 하면 단어를 다 구분할 수 있는 것 같다.)
- ✓ 단순히 모든 토큰을 붙인 후 “_” 캐릭터만 공백으로 바꿔주면 된다.

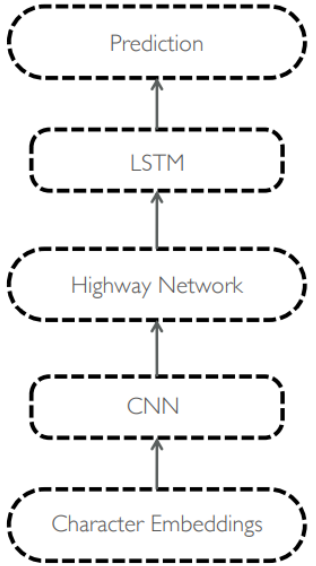
Other Models

- ✓ 요즘은 직전에 설명한 BPE, WPM, SPM, Unigram을 많이 사용한다.
- ✓ 하지만 이전에는 char embedding + conv, char embedding + LSTM, char embedding + CNN + LSTM 등이 있었다(highway net)



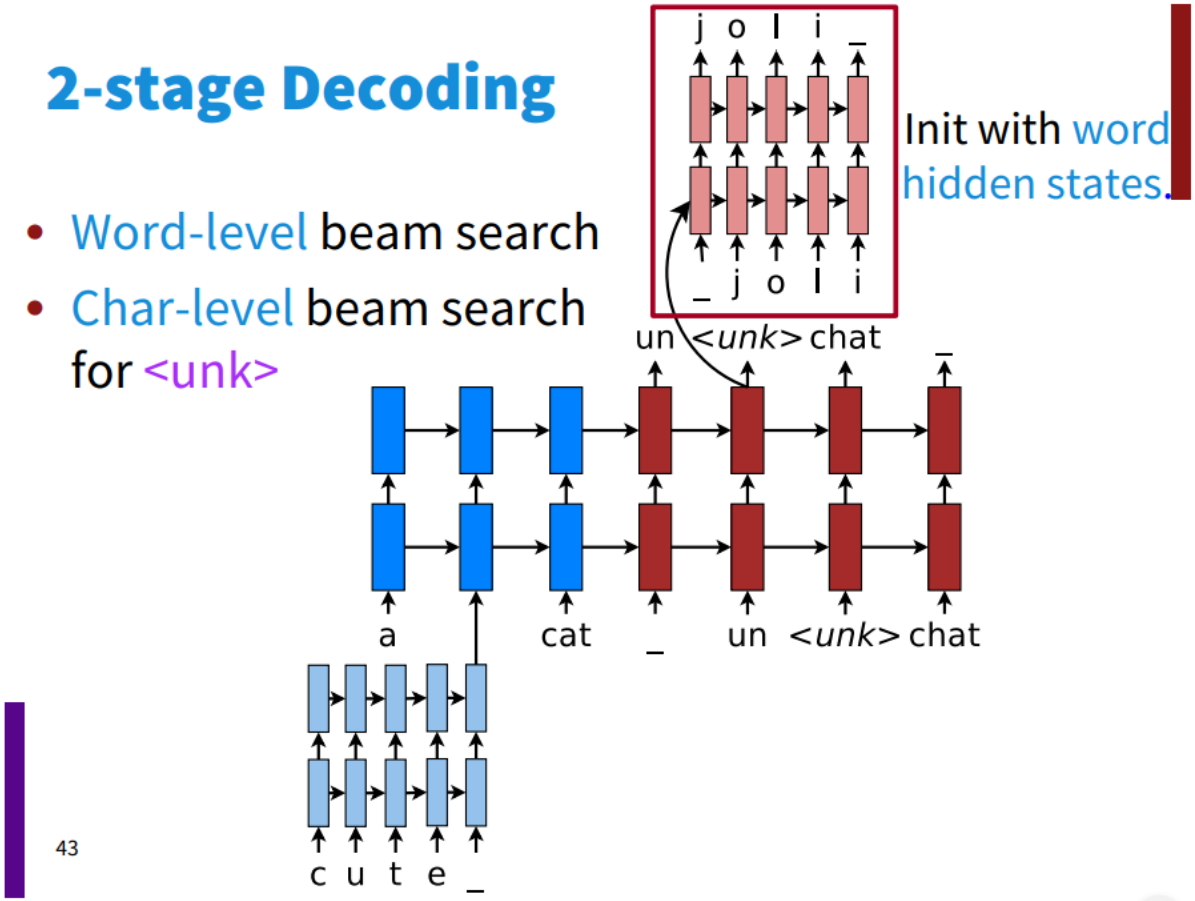






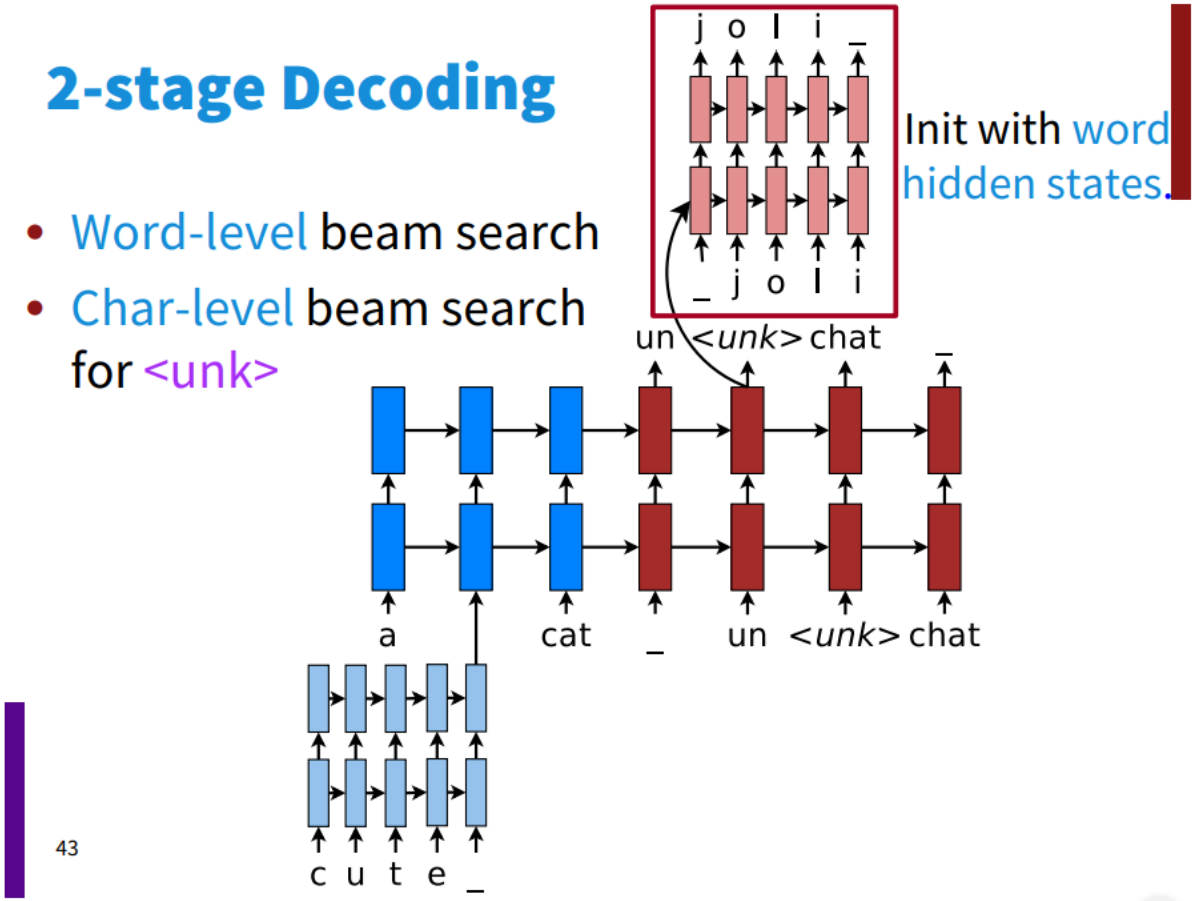
Hybrid NMT

- ✓ BPE는 아니고, word-level과 char-level을 섞어 쓰는 것이다.
- ✓ <UNK>가 나왔을 때 char-level 의 subnet을 이용하는 것이다.



Hybrid NMT

- ✓ BPE는 아니고, word-level과 char-level을 섞어 쓰는 것이다.
- ✓ <UNK>가 나왔을 때 char-level 의 subnet을 이용하는 것이다.



Hybrid NMT

English-Czech Results

- Train on WMT'15 data (12M sentence pairs)
  - newstest2015

Systems	BLEU
Winning WMT'15 (Bojar & Tamchyna, 2015)	18.8
Word-level NMT (Jean et al., 2015)	18.3
Hybrid NMT (Luong & Manning, 2016)*	**20.7**

30x data
3 systems

Large vocab
+ copy mechanism

45

FastText

- ✓ 역시 HybridNMT의 일종이며 word2vec을 기본으로 하되 subword 임베딩 기법을 포함한다.

참고 문헌

한글 블로그 자료

- https://wikidocs.net/22592
- https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/07/06/fasttext/
- https://huffon.github.io/2020/07/05/tokenizers/

CS224n-winter 2019-syllabus-lecture 7

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/

Questions & Answers

Thank you