

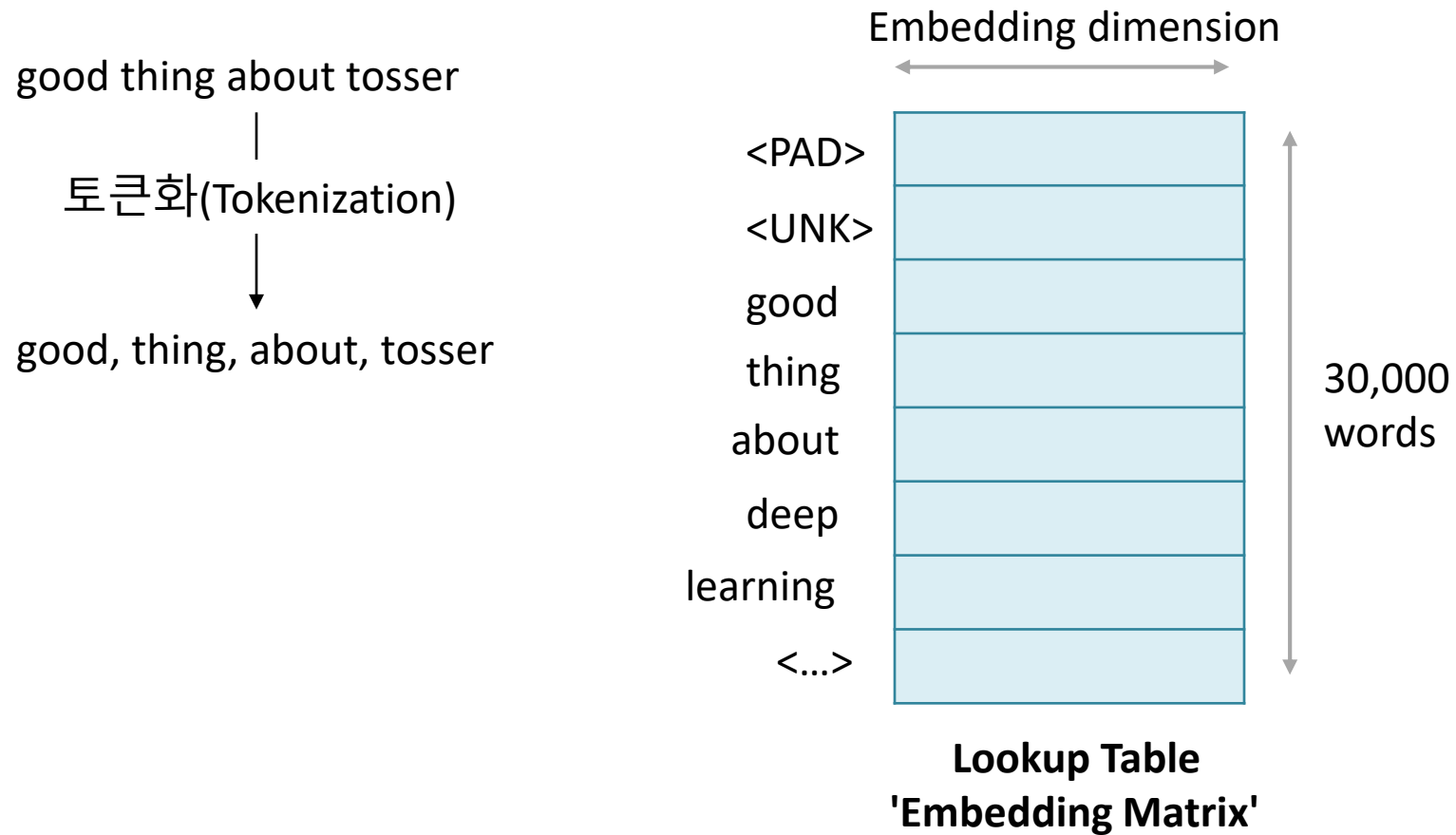
Neural Machine Translation of Rare Words with Subword Units

2019/07/20
유원준

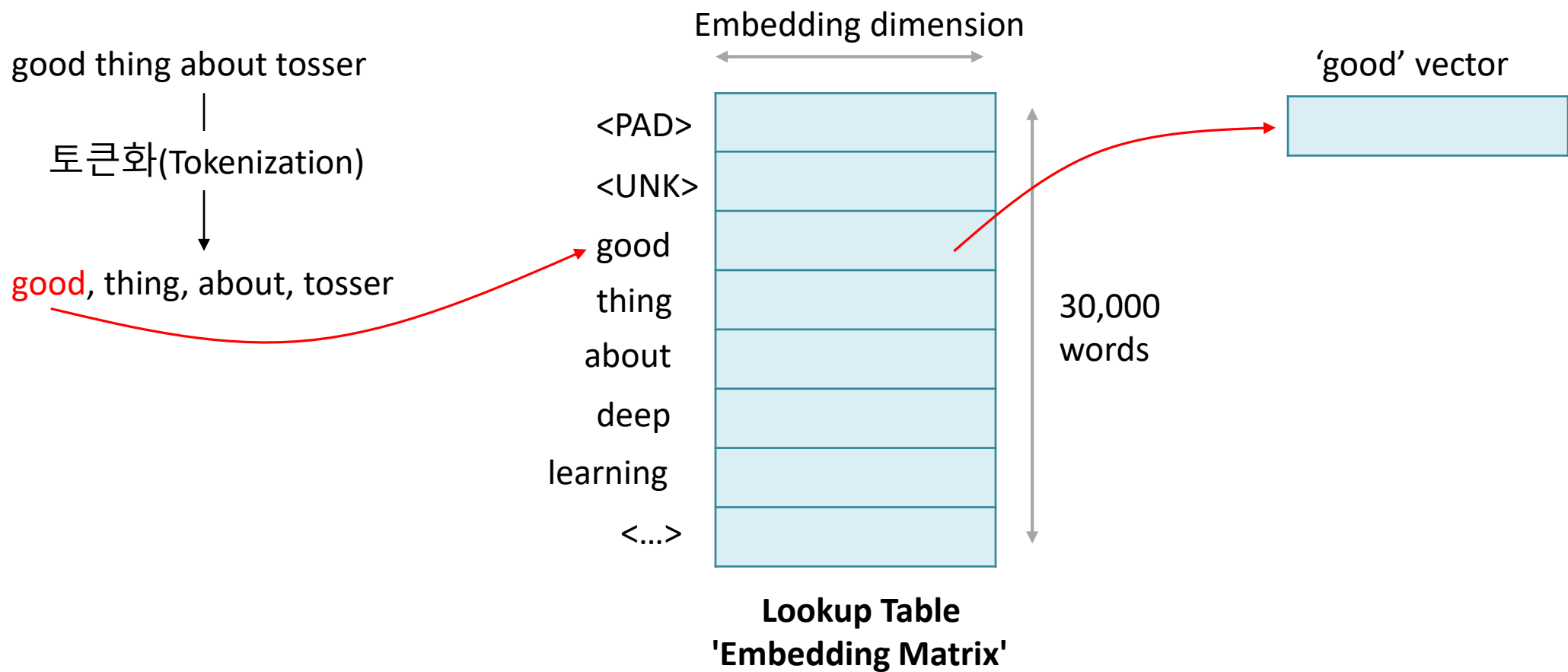
Abstract and Introduction

- NMT 모델은 한정된 크기의 단어 사전(30,000 ~ 50,000)을 가지지만 실제의 단어는 수는 한정되지 않는다.
- 이로 인해 단어 사전에 없는 단어(Out-Of-Vocabulary, OOV) 문제가 발생한다.
- 이 논문은 OOV를 내부단어(subword) 단위(unit)의 인코딩으로 해결할 것을 제안한다..
- 여기서 이를 위해 BPE(Byte Pair Encoding)이라는 알고리즘을 사용한다.

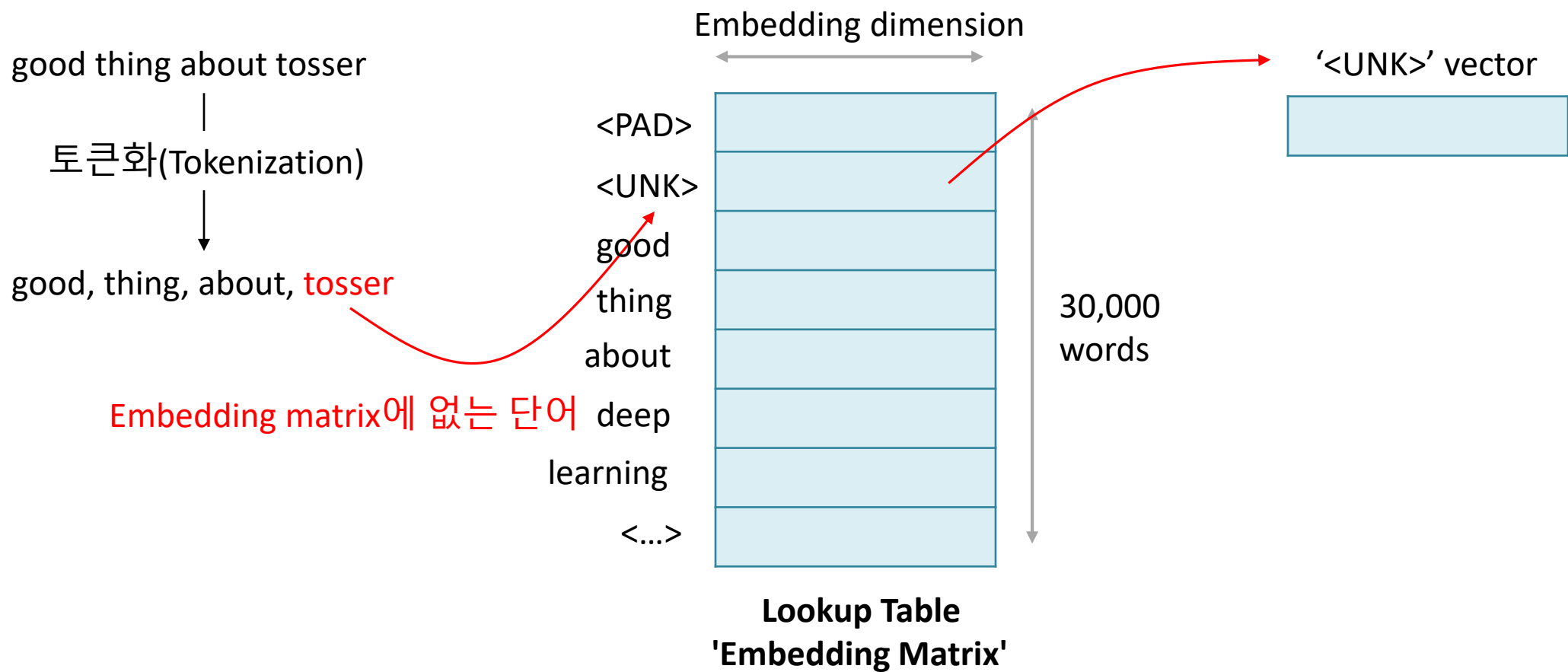
OOV(Out-of-Vocabulary) Problem



OOV(Out-of-Vocabulary) Problem



OOV(Out-of-Vocabulary) Problem



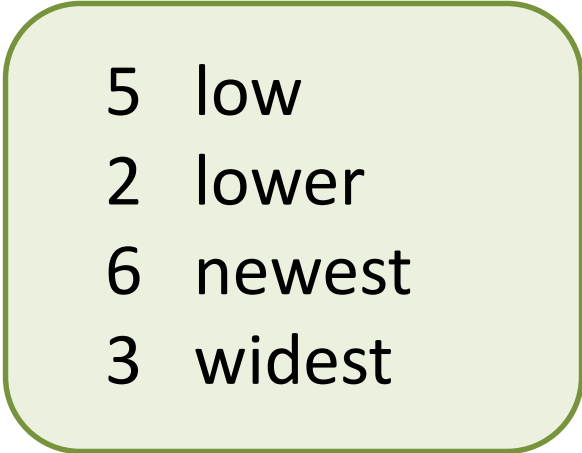
Byte Pair Encoding 개요

- BPE 자체는 1994년에 제안된 데이터 압축 알고리즘
- 자주 등장하는 Byte Pair는 새로운 하나의 Byte가 된다.
- 이를 단어 분리(Word segmentation)에 도입한다.
- Bottom-up 방식의 클러스터링
- 데이터의 모든 글자(character) 단위의 유니그램 단어 사전에서 시작한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.
- (자주 등장하는 바이그램을 하나의 유니그램으로 병합한다.)

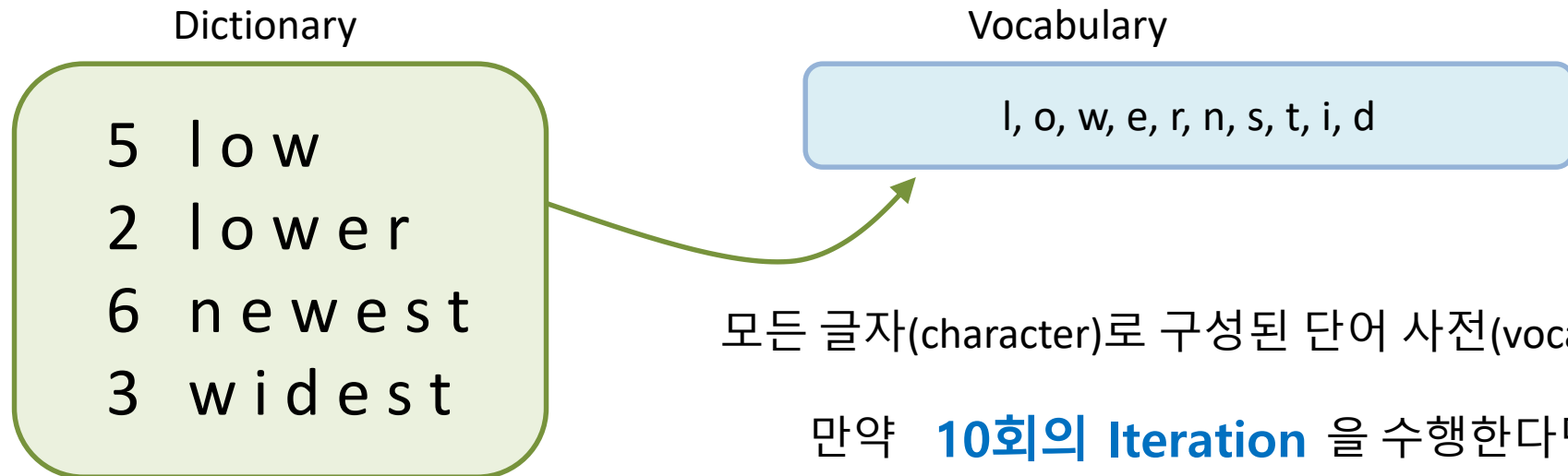
Dictionary



5	low
2	lower
6	newest
3	widest

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다. (자주 등장하는 바이그램을 하나의 유니그램으로 병합한다.)



Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

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

Vocabulary

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

Iteration 1

e, s의 pair는 9의 빈도수를 가진다.
e, s를 es로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

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, i, d, **est**

Iteration 2

es, t의 pair는 9의 빈도수를 가진다.
es, t를 est로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 **lo** w
2 **lo** w e r
6 n e w e s t
3 w i d e s t

Vocabulary

w, e, r, n, i, d, e s t, **lo**

Iteration 3

l, o의 pair는 7의 빈도수를 가진다.
l, o는 lo로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 **low**
2 **low** e r
6 n e w e s t
3 w i d e s t

Vocabulary

w, e, r, n, i, d, e s t, **low**

Iteration 4

lo, w의 pair는 7의 빈도수를 가진다.
lo, w는 low로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 low
2 low e r
6 **ne** w est
3 w i d est

Vocabulary

w, e, r, i, d, est, low, **ne**

Iteration 5

n, e의 pair는 6의 빈도수를 가진다.
n, e는 ne로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 low
2 low e r
6 **new** est
3 w i d est

Vocabulary

w, e, r, i, d, est, low, **new**

Iteration 6

ne, w의 pair는 6의 빈도수를 가진다.
ne, w는 new로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 low
2 low e r
6 **newest**
3 w i d est

Vocabulary

w, e, r, i, d, est, low, **newest**

Iteration 7

new, est의 pair는 6의 빈도수를 가진다.
new, est는 newest로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 low
2 low e r
6 newest
3 **wi** d est

Vocabulary

e, r, d, est, low, newest, **wi**

Iteration 8

w, i의 pair는 3의 빈도수를 가진다.
w, i는 wi로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 low
2 low e r
6 newest
3 **wid** est

Vocabulary

e, r, est, low, newest, **wid**

Iteration 9

wi, d의 pair는 3의 빈도수를 가진다.
wi, d는 wid로 병합한다.

Byte Pair Encoding

- 데이터의 모든 글자(character)의 유니그램 단어 사전에서 시작한다.
- 자주 등장하는 Byte Pair는 새로운 Byte가 된다.

Dictionary

5 low
2 low e r
6 newest
3 **widest**

Vocabulary

e, r, low, newest, **widest**

Iteration 10

wid, est의 pair는 3의 빈도수를 가진다.
wid, est는 widest로 병합한다.

Byte Pair Encoding

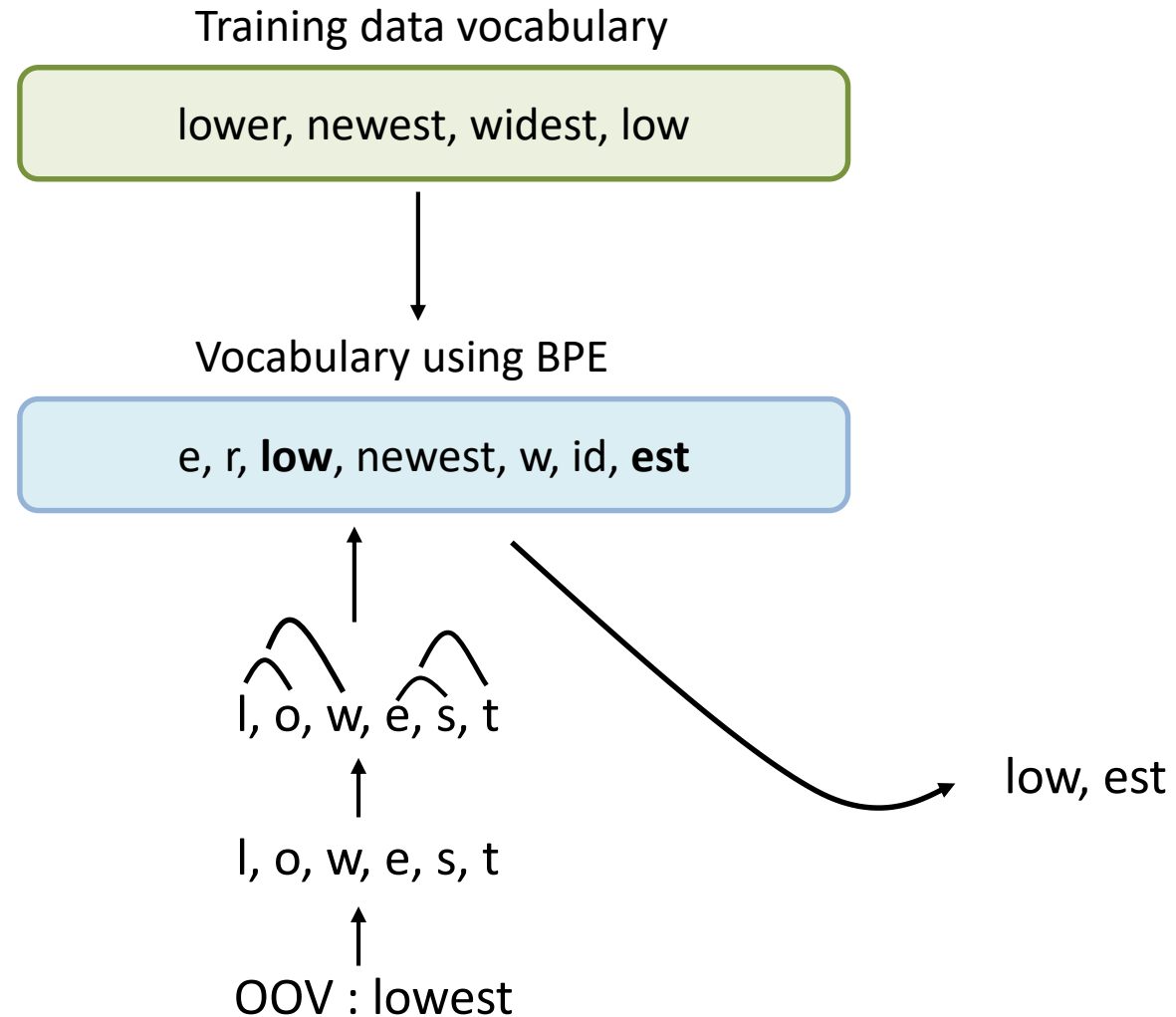
- Iteration을 할 수록 단어 사전의 크기가 커진다. → 원하는 단어 사전 크기 결정 가능
- 글자(character) 단위로 쪼개면 학습이 가능하므로 언어에 종속적이지 않은 알고리즘.

Vocabulary

e, r, low, newest, widest

- 자주 등장하는 단어는 결국 단어 그 자체가 unit이 된다.
- 자주 등장하지 않는 희귀 단어(rare word)는 내부 단어(subword)가 unit이 된다.

OOV(Out-Of-Vocabulary) Problem



BPE Implementation

```
[1] import re, collections

def get_stats(vocab):
    """Compute frequencies of adjacent pairs of symbols."""
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i], symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?!\W\S)' + bigram + r'(!\W\S)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out
```

'low </w>': 5, 'lower </w>': 2,
'newest </w>': 6, 'widest </w>': 3

```
[2] from IPython.display import display, Markdown, Latex

train_data = {'low </w>': 5, 'lower </w>': 2, 'newest </w>': 6, 'widest </w>': 3}

bpe_codes = {}
bpe_codes_reverse = {}

num_merges = 10

for i in range(num_merges):
    display(Markdown("### Iteration {}".format(i + 1)))
    pairs = get_stats(train_data)
    best = max(pairs, key=pairs.get)
    train_data = merge_vocab(best, train_data)

    bpe_codes[best] = i
    bpe_codes_reverse[best[0] + best[1]] = best

    print("new merge: {}".format(best))
    print("train data: {}".format(train_data))
```

Implementation available on Colab:

<https://colab.research.google.com/drive/1G9vRvOThc5We0ji-x-aNU4CoeOVu3fV->

BPE Implementation

```
[1] import re, collections

def get_stats(vocab):
    """Compute frequencies of adjacent pairs of symbols."""
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i], symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?!\W)' + bigram + r'(!\W)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out
```

```
[2] from IPython.display import display, Markdown, Latex

train_data = {'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3}

bpe_codes = {}
bpe_codes_reverse = {}

num_merges = 10

for i in range(num_merges):
    display(Markdown("## Iteration {}".format(i + 1)))
    pairs = get_stats(train_data)
    best = max(pairs, key=pairs.get)
    train_data = merge_vocab(best, train_data)

    bpe_codes[best] = i
    bpe_codes_reverse[best[0] + best[1]] = best

    print("new merge: {}".format(best))
    print("train data: {}".format(train_data))
```

chracter 단위로 분할

끝을 의미하는 특수기호

빈도수

'l o w </w>': 5, 'l o w e r </w>': 2,
'n e w e s t </w>': 6, 'w i d e s t </w>': 3

1 iteration: ('e', 's')
2 iteration: ('es', 't')
3 iteration: ('est', '</w>')
4 iteration: ('l', 'o')
5 iteration: ('lo', 'w')
6 iteration: ('n', 'e')
7 iteration: ('ne', 'w')
8 iteration: ('new', 'est</w>')
9 iteration: ('low', '</w>')
10 iteration: ('w', 'i')

'low</w>': 5, 'low e r </w>': 2,
'newest</w>': 6, 'wi d est</w>': 3

Implementation available on Colab:

<https://colab.research.google.com/drive/1G9vRvOThc5We0ji-x-aNU4CoeOVu3fV->

BPE Implementation

```
[1] import re, collections

def get_stats(vocab):
    """Compute frequencies of adjacent pairs of symbols."""
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i], symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?!\W)' + bigram + r'(!\W)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out
```

```
[2] from IPython.display import display, Markdown, Latex

train_data = {'low </w>': 5, 'low er </w>': 2, 'newest </w>': 6, 'wid est </w>': 3}

bpe_codes = {}
bpe_codes_reverse = {}
num_merges = 10

for i in range(num_merges):
    display(Markdown("## Iteration {}".format(i + 1)))
    pairs = get_stats(train_data)
    best = max(pairs, key=pairs.get)
    train_data = merge_vocab(best, train_data)

    bpe_codes[best] = i
    bpe_codes_reverse[best[0] + best[1]] = best

    print("new merge: {}".format(best))
    print("train data: {}".format(train_data))
```

'low</w>': 5, 'low e r </w>': 2,
'newest</w>': 6, 'wi d est</w>': 3



총 9개의 내부단어(subword) unit

'low</w>': 5,
'low': 2,
'e': 2,
'r': 2,
'</w>': 2,
'newest</w>': 6,
'wi': 3,
'd': 3,
'est</w>': 3

Implementation available on Colab:

<https://colab.research.google.com/drive/1G9vRvOThc5We0ji-x-aNU4CoeOVu3fV->

BPE Implementation - OOV

Training data vocabulary

lower, newest, widest, low

Subword unit

'low</w>': 5,
'low': 2,
'e': 2,
'r': 2,
'</w>': 2,
'newest</w>': 6,
'wi': 3,
'd': 3,
'est</w>': 3

OOV : 'lowest' ?

Implementation available on Colab:

<https://colab.research.google.com/drive/1G9vRvOThc5We0ji-x-aNU4CoeOVu3fV->

BPE Implementation - OOV

```
def get_pairs(word):
    """Return set of symbol pairs in a word.
    Word is represented as tuple of symbols (symbols being variable-length strings)."""
    pairs = set()
    prev_char = word[0]
    for char in word[1:]:
        pairs.add((prev_char, char))
        prev_char = char
    return pairs

def encode(orig):
    """Encode word based on list of BPE merge operations, which are applied consecutively"""
    word = tuple(orig) + ('</w>')
    display(Markdown("__word split into characters: __<tt>{}</tt>".format(word)))
    pairs = get_pairs(word)

    if not pairs:
        return orig

    iteration = 0
    while True:
        iteration += 1
        display(Markdown("__Iteration {}: __".format(iteration)))

        print("bigrams in the word: {}".format(pairs))
        bigram = min(pairs, key = lambda pair: bpe_codes.get(pair, float('inf')))
        print("candidate for merging: {}".format(bigram))
        if bigram not in bpe_codes:
            display(Markdown("__Candidate not in BPE merges, algorithm stops..."))
            break
        first, second = bigram
        new_word = []
        i = 0
        while i < len(word):
            try:
                j = word.index(first, i)
                new_word.extend(word[i:j])
                i = j
            except:
                new_word.extend(word[i:])
                break

            if word[i] == first and i < len(word)-1 and word[i+1] == second:
                new_word.append(first+second)
                i += 2
            else:
                new_word.append(word[i])
                i += 1
        new_word = tuple(new_word)
        word = new_word
        print("word after merging: {}".format(word))
        if len(word) == 1:
            break
        else:
            pairs = get_pairs(word)

    # don't print end-of-word symbols
    if word[-1] == '</w>':
        word = word[:-1]
    elif word[-1].endswith('</w>'):
        word = word[:-1] + (word[-1].replace('</w>', ''),)

    return word
```

OOV : 'lowest'

encode("lowest")

l o w e s t </w>

1 iteration: ('l', 'o', 'w', 'es', 't', '<\w>')
2 iteration: ('l', 'o', 'w', 'est', '<\w>')
3 iteration: ('l', 'o', 'w', 'est<\w>')
4 iteration: ('lo', 'w', 'est<\w>')
5 iteration: ('low', 'est<\w>')
Result : ('low', 'est')

learned subword unit

'low</w>': 5,
'low': 2,
'e': 2,
'r': 2,
'</w>': 2,
'newest</w>': 6,
'wi': 3,
'd': 3,
'est</w>': 3

Implementation available on Colab:

<https://colab.research.google.com/drive/1G9vRvOThc5We0ji-x-aNU4CoeOVu3fV->

Results

segmentation	# tokens	# types	# UNK
none	100 m	1 750 000	1079
characters	550 m	3000	0
character bigrams	306 m	20 000	34
character trigrams	214 m	120 000	59
compound splitting [△]	102 m	1 100 000	643
morfeessor*	109 m	544 000	237
hyphenation [◇]	186 m	404 000	230
BPE	112 m	63 000	0
BPE (joint)	111 m	82 000	32
character bigrams (shortlist: 50 000)	129 m	69 000	34

BPE : learning two independent encodings,
one for the source,
one for the target vocabulary

BPE(joint) : learning the encoding on the
union of the two vocabularies

Table 1: Corpus statistics for German training corpus with different word segmentation techniques. #UNK: number of unknown tokens in newstest2013. [△]: (Koehn and Knight, 2003); *: (Creutz and Lagus, 2002); [◇]: (Liang, 1983).

Results

name	segmentation	shortlist	vocabulary		BLEU		CHRF3		unigram F ₁ (%)		
			source	target	single	ens-8	single	ens-8	all	rare	OOV
syntax-based (Sennrich and Haddow, 2015)					24.4	-	55.3	-	59.1	46.0	37.7
WUnk	-	-	300 000	500 000	20.6	22.8	47.2	48.9	56.7	20.4	0.0
WDict	-	-	300 000	500 000	22.0	24.2	50.5	52.4	58.1	36.8	36.8
C2-50k	char-bigram	50 000	60 000	60 000	22.8	25.3	51.9	53.5	58.4	40.5	30.9
BPE-60k	BPE	-	60 000	60 000	21.5	24.5	52.0	53.9	58.4	40.9	29.3
BPE-J90k	BPE (joint)	-	90 000	90 000	22.8	24.7	51.7	54.1	58.5	41.8	33.6

Table 2: English→German translation performance (BLEU, CHRF3 and unigram F₁) on newstest2015. Ens-8: ensemble of 8 models. Best NMT system in bold. Unigram F₁ (with ensembles) is computed for all words ($n = 44085$), rare words (not among top 50 000 in training set; $n = 2900$), and OOVs (not in training set; $n = 1168$).

References

- Neural Machine Translation of Rare Words with Subword Units (<https://arxiv.org/abs/1508.07909>)
- <https://www.youtube.com/watch?v=bv9HUf-tGWo>
- http://ufal.mff.cuni.cz/~helcl/courses/npfl116/ipython/byte_pair_encoding.html
- CS224N: Lecture 12: Information from parts of words: Subword Models
- <https://lovit.github.io/nlp/2018/04/02/wpm/>