

KUBIG 2023-1

겨울방학 자연어처리(NLP) 분반

WEEK 3



Part 1

복습과제 리뷰

- 전처리 전후 모델 결과 비교
- 유사도 기반 음식점 추천 시스템

```
▶ ## 문자열 처리 함수
def review_to_words(raw_review):
    letters_only = re.sub('[^ㄱ-ㅎ가-힣0-9]', ' ', raw_review)
    words = letters_only.split()
    stops = set(stopword)
    meaningful_words = [w for w in words if not w in stops]
    return(' '.join(meaningful_words))
```

```
def Int_Preprocess(sent) :

    step1 = re.sub('[^가-힣]', ' ', sent)

    step2 = emoticon_normalize(step1, num_repeats=2)

    step3 = repeat_normalize(step2, num_repeats=2)

    token = l_tokenizer.tokenize(step3, flatten=False)
    step4 = []
    for i in range(len(token)):
        step4.append(token[i][0])

    step5 = [word for word in step4 if not word in stop_words]

    text = " ".join(step5)

    return text
```

```
▶ okt = Okt()
spacing = Spacing()

def preprocess(text):
    text = re.sub('[^가-힣]', ' ', text) # 한글 제외하고 모두 제거
    text = spacing(text)
    text = okt.morphs(text, stem=True) # Okt 활용하여 형태소 단위로 토큰화
    text = [x for x in text if x not in stopwords] # 불용어 제거
    text = " ".join(text) # 토큰화된 것 하나의 문장으로 이어주기
    return text
```

```
[ ] def preprocessing(data):
    preprocessed = []
    for i in data:
        cleaned = re.sub('[^ㄱ-ㅎ가-힣0-9()#~]', ' ', i)
        morphs = okt.morphs(cleaned, stem=True)
        word = [word for word in morphs if word not in stopwords]
        word = " ".join(word)
        preprocessed.append(word)
    return preprocessed
```

```
[ ] for i in range(len(df)):
    example_df=df['text'].iloc[i]
    tokenized_df = okt.morphs(example_df) # okt 토큰화
    result_df = [word for word in tokenized_df if not word in stopwords] # 불용어 제거
    join_df = " ".join(result_df)
    df['text'].iloc[i] = join_df
```

전처리 check

라이브러리에서 제공하는 모델의 기본 input을 보면 불용어제거와 같은 전처리를 진행하지 않은 문자열 데이터이다.

1. html 태그가 있는가 -> no
2. 언어를 통일할 필요가 있는가 -> 최종 목적은 요약이기 때문에 숫자, 한글, 영어를 모두 살려야 할 듯함(혹시 고유명사인 핵심단어 영어가 있다면 지워질 가능성 있음)
3. 토큰화가 필요한가 -> no(모델의 영역)
4. 불용어 처리 -> no(모델의 영역)
5. 어간추출 or 음소표기법 -> no(모델의 영역)
6. 그 외 체크사항 : 엔터 키(\n)가 지워졌는지 확인할 필요 있음

비교를 위해 정규표현식만 사용 -> 불용어처리까지 사용을 분리해서 데이터를 저장해본다

```
[ ] import re
letters_only = re.sub('#n', '', dat['words'][0]) #줄바꿈 표시 제거
letters_only = re.sub('[^a-zA-Z0-9가-힣.,%~ ]', '', letters_only)
#특수 문자 중 -, %, ~, ,, . 등은 유의미한 뜻을 지니므로 제거하지 않음
#특수문자 -는 괄호 짝 앞에 써줘야 오류가 나지 않음!!!
#ㄱ-ㅎ는 제거함(ㅠ 같은 표현 존재)
letters_only
```

주석

파이썬에서 list는 강력한 자료구조이지만 구조적으로 느리기 때문에 만약 자료형이 통일된 현재 데이터들과 같은 자료들을 사용하며 시간의 단축이 필요하다면 numpy의 array를 사용했을 때 탐색 시간이 줄어들 것이다

주석

bertshared-kor-base model 내에는 자체적으로 한국어 text에 대한 전처리 기능이 존재하기 때문에 전처리되지 않은 데이터를 넣었을 때 더 강력한 기능을 하는듯해 보인다.

따라서 해당 데이터같은 경우 전처리를 할 수록 성능이 안좋아지는 것으로 보인다.

(데이터 전처리 과정에서 Okt의 기능 중 어간 추출을 활용하다보니 과거, 현재, 미래의 시제가 사라졌는데 모델에서는 해당 시제까지 활용할 수도 있음)

또한 summarize 메소드는 return 함수가 아닌 print함수이기 때문에 별도 저장이 안되는 것 같다

민규 님



- 전혀 전처리를 거치지 않으면, 다소 혼란스러운 또는 조잡한 문장이 나오게 되어 요약문으로서의 성격에 맞지 않음
- 명사만을 추출해서 요약하면, 의미를 가진 연결어들(형태소 단위)이 모두 빠지게 되어 다소 불완전한 요약 내용 생성되는듯함
- 정규표현식으로 간단한 전처리만 거치더라도 가장 핵심적인 문장의 주제는 포함하는 것을 확인할 수 있었음
- 문장 길이에 따라 summarize 모델이 돌아가지 않는 경우 빈번했음. 이에 대한 디버깅을 어떻게 하는 것인지?

윤기 님

>>>>>>>>> 19 비교 <<<<<<<<<<

***** 전처리 전 *****

건설업을 하려는 자는 기술능력 자본금 시설 등 세 가지 부문에서 일정한 등록기준을 갖추어야 하며, 일 가정 양립을 지원하는 조직문화를 확립하기 위해 7월 27일부터 시행되는 건설산업기본법 시행령 일부개정 관련 내용이 포함된다.

None

***** 전처리 후 *****

건설산업기본법 시행령 일부개정 관련 내용이 확정된 가운데 일 가정 양립을 지원하는 조직문화 확립을 위해 육아 근로시간 단축근무 중인 건설 등록기준 기술능력으로 인정 필요 필요하면 행정처분 대상에서 제외하는 등의 제도를 도입할 예정이다.

None

>>>>>>>>> 20 비교 <<<<<<<<<<

***** 전처리 전 *****

우리 나라 지금 우리나라는 지금 나라의 모든 가능성을 불태워서 문화승리 기술을 연마하고 당장 생존을 위해 무언가를 하는 경향이 있는데 비교와 경쟁의 부작용이 가장 두드러지는 요즘이다.

None

***** 전처리 후 *****

결혼 하락과 출산율 하락으로 인한 인구수 감소로 현재 문화 기술승리 중인 저쪽인 저쪽은 노령인구가 돈이 있어서 버티는 생존을 위해 계속 드라마를 만들 수밖에 없다.

None

태영 님

▼ 성능 확인 결과

1. 어간 추출 제외 전처리
2. 전처리 x
3. 어간 추출 포함 전처리

순으로 성능이 우수함을 확인할 수 있음

[] summarize(data['description'][0][:512])

한국부동산원에 따르면 지난해 11월 서울 아파트 실거래가는 전달 대비 6.47 % 하락하면서 12월 잠정치를 반영하면 2022년 한 해 수도권 아파트 매매 가격은 20 % 이상 폭락할 것으로 전망된다.

[] summarize(data['preprocessed2'][0][:532])

한국 부동산 원 조사 시작 2006년 2월 조사 이래 가장 많이 하락한 것으로 조사됐으며 이는 2008년 글로벌 금융위기 이후 가장 큰 폭락 폭이라고 한다.

기영 님

```
[ ] idx = id2res['만가타']
sim_scores = [(i, c) for i, c in enumerate(cosine_matrix[idx]) if i != idx] # 자기 자:
sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse=True) # 유사도가 높은 순
```

```
▶ sim_scores = [(res2id[i], score) for i, score in sim_scores[0:10]]
sim_scores
```

```
⦿ [('파씨오네', 0.48974293394941737),
   ('BISTROT de YOUNTVILLE', 0.4738733555347193),
   ('알라프리마', 0.4538564558347222),
   ('윤', 0.44597587415083406),
   ('무오키', 0.44292525492725354),
   ('가디록', 0.4359932902295078),
   ('SOOT (휴업중)', 0.43156981891947543),
   ('톡톡', 0.4284617699832948),
   ('세상의모든아침', 0.42688631449360454),
   ('라미띠에', 0.4223356876868435)]
```

7. 느낀 것

- 어느 식당은 리뷰가 수백 개에 달하고 어느 식당은 10개 미만이다. 식당마다 리뷰 수가 다른 것을 어떻게 보정해줄지 고민해볼 만하다. 식당마다 리뷰 개수를 통일하는 방법도 있고, 통일한다면 몇 개로 할지도 고민해봐야 한다.
- 리뷰 뿐만이 아니라 평점, 위치, 음식 종류에 따른 유사도도 추가해야 제대로 된 추천 시스템이 구현될 것 같다.
- 다만 여러 유사도를 결국 하나로 합치는 과정이 필요할 것 같은데, 각각에 얼마나 큰 가중치를 부여하여 합칠 것인지 명확한 기준을 세우기가 어려울 것 같다.
- 위치 기반 유사도를 판단하는 것은 위도,경도 데이터를 불러와서 잘 매핑해보면 될 것 같은데, 나중에 구현해보면 재밌을 것 같다.
- 실제로 쓰이는 식당 추천 알고리즘 원리들을 엿볼 수 있다면 흥미로울 것 같다.

희준 님

```
[ ] idx = id2res['만가타']
sim_scores = [(i, c) for i, c in enumerate(cosine_matrix[idx]) if i != idx] # 자기 자:
sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse=True) # 유사도가 높은 순
```

```
▶ sim_scores = [(res2id[i], score) for i, score in sim_scores[0:10]]
sim_scores
```

```
⦿ [('파씨오네', 0.48974293394941737),
   ('BISTROT de YOUNTVILLE', 0.4738733555347193),
   ('알라프리마', 0.4538564558347222),
   ('윤', 0.44597587415083406),
   ('무오키', 0.44292525492725354),
   ('가디록', 0.4359932902295078),
   ('SOOT (휴업중)', 0.43156981891947543),
   ('톡톡', 0.4284617699832948),
   ('세상의모든아침', 0.42688631449360454),
   ('라미띠에', 0.4223356876868435)]
```

7. 느낀 것

- 어느 식당은 리뷰가 수백 개에 달하고 어느 식당은 10개 미만이다. 식당마다 리뷰 수가 다른 것을 어떻게 보정해줄지 고민해볼 만하다. 식당마다 리뷰 개수를 통일하는 방법도 있고, 통일한다면 몇 개로 할지도 고민해봐야 한다.
- 리뷰 뿐만이 아니라 평점, 위치, 음식 종류에 따른 유사도도 추가해야 제대로 된 추천 시스템이 구현될 것 같다.
- 다만 여러 유사도를 결국 하나로 합치는 과정이 필요할 것 같은데, 각각에 얼마나 큰 가중치를 부여하여 합칠 것인지 명확한 기준을 세우기가 어려울 것 같다.
- 위치 기반 유사도를 판단하는 것은 위도,경도 데이터를 불러와서 잘 매핑해보면 될 것 같은데, 나중에 구현해보면 재밌을 것 같다.
- 실제로 쓰이는 식당 추천 알고리즘 원리들을 엿볼 수 있다면 흥미로울 것 같다.

희준 님

주석

tqdm : tqdm은 반복문의 현재 진행 상황을 progress bar로 알려주는 모듈이다.

사용법 예시 for i in tqdm(range(10)):

```
nass
```

주석


pe.read_csv의 option 중 low_memory = False는 대용량의 데이터를 불러올 때 데이터 타입을 확인하는 과정을 생략해 시간 단축 및 오류 감소에 도움을 준다

```
recommend_rest = []
for rest, score in sim_scores_fn:
    recommend_rest.append(rest)
recommend_rest
```

```
[ '라우센트',
  '패션파이버',
  '패션5라폴리에',
  '팬케이크상',
  '패션5라폴리에',
  '조선엘리더부티크',
  '조선엘리더부티크',
  '리틀빅토리',
  '리틀빅토리',
  '패션파이버',
  '센터커피',
  '센터커피',
  '센터커피',
  '리틀빅토리',
  '리틀빅토리',
  '리틀빅토리',
  '멜그쇼즈',
  '리틀빅토리',
  '리틀빅토리',
  '라우센트',
  '라우센트',
  '연남장',
  '패션파이버',
  '패션파이버',
  '리틀빅토리',
  '리틀빅토리',
  '멜그쇼즈',
  '멜그쇼즈',
  '라우센트',
  'Scoff',
  '내자상회']
```

```
[ ] # 추천
from collections import Counter
Counter(recommend_rest).most_common(5)
```

```
[('리틀빅토리', 8), ('라우센트', 4), ('패션파이버', 4), ('센터커피', 3), ('멜그쇼즈', 3)]
```

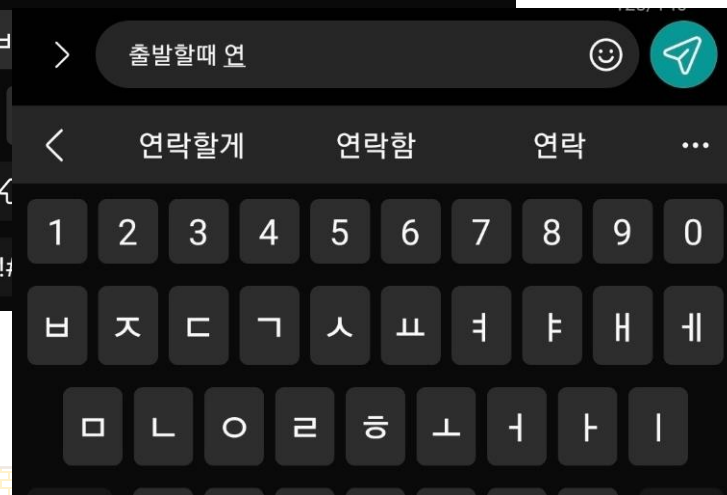
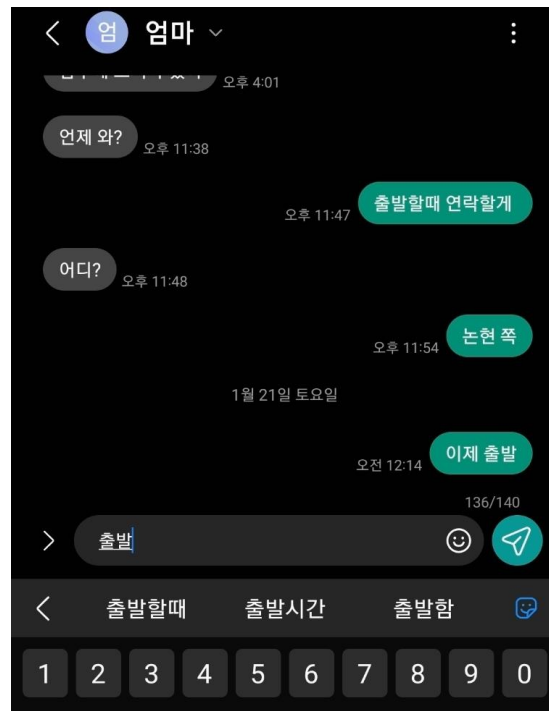
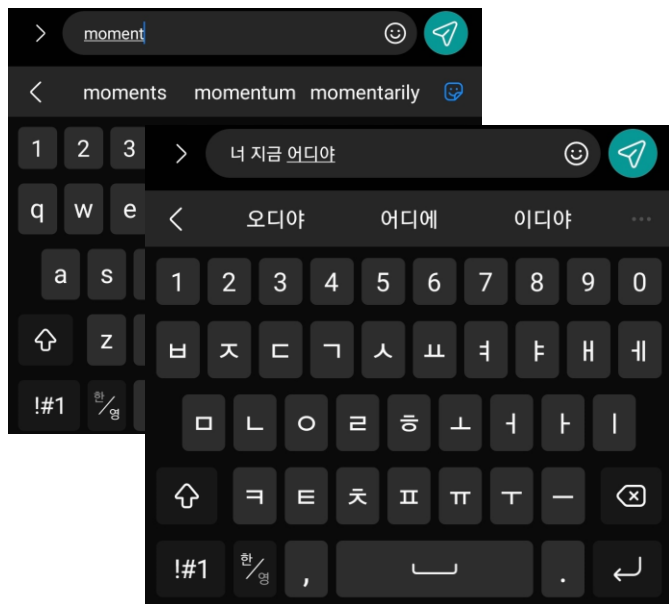



Part 2

3주차진도(1)

언어모델

언어 모델



언어 모델

I 언어 모델 Language Model

: 언어를 모사하기 위해 다음 단어(문장)을 예측하는 모델

- 이전 단어들이 주어졌을 때 다음 단어를 예측
- 양쪽의 단어들이 주어지고 가운데 비어있는 단어를 예측 (BERT)

**통계적 언어 모델
SLM**

: 다양한 말뭉치(corpus)를 만들어 그 안에서 다음 단어를 예측

**인공 신경망을 이용한 언어모델
NNLM**

: 단어 하나를 주고 다음 단어를 예측하도록 학습
or 문장 내 빈칸에 맞추도록 학습

언어 모델

I 통계적 언어 모델 Statistical Language Model

- ▶ 분포 가설 (Distributional Hypothesis)
 ⇒ 단어의 의미는 주변 단어에 의해 형성된다

단어 자체에는 의미가 없고 그 단어가 사용된 문맥이 의미를 형성한다.

조건부 확률을 통해 문맥 안에서 다음 단어를 예측
 문장의 확률은 각 단어들이 이전 단어가 주어졌을 때 다음 단어로 등장할 확률의 곱으로 구성됨

$$P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = \prod_{n=1}^n P(w_n | w_1, \dots, w_{n-1})$$

P(With great power comes great responsibility)

= P(With) X P(great | With) X P(power | With great) ⋯ X P(responsibility | With great power comes great)

ex) P(responsibility | With great power comes great)

=

언어 모델

I 통계적 언어 모델 Statistical Language Model

희소 문제 (Sparsity Problem)

: 예측 확률이 코퍼스가 갖고 있는 데이터의 양에 의존함 ~ 수많은 방대한 양의 데이터 필요

▷ 대안1) 상호 정보량 (Pointwise Mutual Information)

"the car..." vs "car drive"

둘의 의미적인 관계를 따져보면 후자가 더 서로 관련이 깊지만, 등장 횟수만을 고려하면 전자가 코퍼스 내에서 많이 마주치게 된다 (the가 고빈도 단어이기 때문)

$$\text{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

+) PPMI = max(0, PMI) 동시발생 횟수 0 일때 음의 무한대 가는거 막고자

언어 모델

▶ 대안 2) N-gram 언어 모델

: 이전에 등장한 모든 단어를 고려하지 않고 일부 단어를 고려 (임의의 개수 n 개 만큼)

$P(\text{responsibility} \mid \text{With great power comes great})$

$\approx P(\text{responsibility} \mid \text{great}) \rightarrow$ 너무 집약

$\approx P(\text{responsibility} \mid \text{comes great})$

코퍼스에서 해당 단어의 시퀀스를 카운트할 확률 높아져

unigrams	With, great, power, comes, great, responsibility
bigrams	With great, great power, power comes, comes great, great responsibility
trigrams	With great power, great power comes, power comes great, comes great responsibility
4-grams	With great power comes, great power comes great, power comes great responsibility

$n=4$ (4-gram)

~~With great~~ power comes great _____

$P(w \mid \text{power comes great}) = \text{count}(\text{power comes great } w) / \text{count}(\text{power comes great})$

그러나 여전히

△희소 문제 : 코퍼스에서 카운트할 수 있는 확률 자체는 높이지만 여전히 sparse 함

△ n 을 선택하는 trade-off 문제

: n 을 크기 - 희소문제 커짐. 모델 사이즈 커짐

: n 을 작게 - 카운트는 잘 됨. 근사의 정확도는 떨어짐 ($n \leq 5$)

성능 지표

| PPL (Perplexity)

- : 이전 단어에서 다음 단어를 예측할 때 몇 개의 단어 후보를 고려하는지를 알려주는 지표
- : 고려해야하는 가짓수 (branching factor) 가 많을수록 bad

$$PPL(W) = P(w_1, w_2, w_3, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \dots, w_N)}}$$

| BLEU (Bilingual Evaluation Understudy Score)

- N-gram overlap between machine translation output and reference translation
- Compute precision for n-grams of size 1 to 4
- Add brevity penalty (for too short translations)

$$BLEU = \min\left(1, \frac{\text{output-length}}{\text{reference-length}}\right) \left(\prod_{i=1}^4 \text{precision}_i\right)^{\frac{1}{4}}$$

- : n gram 을 통해서 실제값과 예측값의 순서쌍이 얼마나 겹치는지
- + 추가로 중복 or 문장 길이에 대한 보정을 할 수 있다 (Clipping, Brevity Penalty)

- 예측된 sentence : 빛이 썬는 노인은 완벽한 어두운곳에서 잠든 사람과 비교할 때 강박증이 심해 질 기회가 훨씬 높았다
- true sentence : 빛이 썬는 사람은 완벽한 어둠에서 잠든 사람과 비교할 때 우울증이 심해질 가능성이 훨씬 높았다

- 1-gram precision: $\frac{\text{일치하는 1-gram의 수(예측된 sentence중에서)}}{\text{모든 1-gram쌍 (예측된 sentence중에서)}} = \frac{10}{14}$
- 2-gram precision: $\frac{\text{일치하는 2-gram의 수(예측된 sentence중에서)}}{\text{모든 2-gram쌍 (예측된 sentence중에서)}} = \frac{5}{13}$
- 3-gram precision: $\frac{\text{일치하는 3-gram의 수(예측된 sentence중에서)}}{\text{모든 3-gram쌍 (예측된 sentence중에서)}} = \frac{2}{12}$
- 4-gram precision: $\frac{\text{일치하는 4-gram의 수(예측된 sentence중에서)}}{\text{모든 4-gram쌍 (예측된 sentence중에서)}} = \frac{1}{11}$

$$\left(\prod_{i=1}^4 \text{precision}_i\right)^{\frac{1}{4}} = \left(\frac{10}{14} \times \frac{5}{13} \times \frac{2}{12} \times \frac{1}{11}\right)^{\frac{1}{4}}$$

성능 지표

I ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

정답문장: "한화는 10 년 안에 우승 할 것이다."

생성문장: "두산은 3 년 안에 우승 할 것이다."

: n gram Recall

+ 여기에 다양한 n gram을 적용할 수 있고, 가장 긴 시퀀스, 연속적인 매칭에 가중치 등 여러 variation 가능

$$N_{\text{정답문장}} = 7$$

$$N_{\text{년,안,우승,할,것이다}} = 5$$

$$ROUGE-1 = \frac{5}{7}$$

I Confusion Matrix

Accuracy / Precision / Recall / F1 Score

		ACTUAL	
		Negative	Positive
PREDICTION	Negative	TRUE NEGATIVE	FALSE NEGATIVE
	Positive	FALSE POSITIVE	TRUE POSITIVE

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$specificity = \frac{TN}{TN + FP}$$

성능 지표


Task Acronym	Task Name	Dataset for task	Expected output	Evaluation criteria
BoolQ	Boolean Questions	A QA task consisting of a short passage and a yes/no question about the same.	Predict the correct answer of the question.	Accuracy.
CB	Commitment Bank	A collection of short texts in which at least one sentence contains an embedded clause. Each example consists of a premise containing an embedded clause and corresponding hypothesis is the extraction of that clause,	Three-class textual entailment.	Accuracy and F1 score.
COPA	Choice of Plausible Alternatives	Causal reasoning task.	A premise sentence is given, system must determine either the cause or the effect of the premise from two possible choices.	Accuracy.
MultiRC	Multi-Sentence Reading Comprehension	A QA task, each example consisting of a context paragraph, a question about that paragraph, and a list of possible answers.	Predict which answers are true and which are false. Each question can have multiple possible correct answers.	F1 over all answer-options (F1a) and exact match of each question's set of answers (EM).
ReCoRD	Reading Comprehension with Commonsense Reasoning Dataset	Multiple-choice QA task. Each example consists of a news article and a Cloze-style question about the article in which one entity is masked out.	Predict the masked out entity from a given list of possible entities in the provided passage, where the same entity may be expressed using multiple different surface forms, all of which are considered correct.	Max (over all mentions) token-level F1 and exact match (EM).
RTE	Recognizing Textual Entailment	Same as GLUE.		Accuracy.
WIC	Word-in-Context	Word sense disambiguation task cast as binary classification of sentence pairs. Given two text snippets and a polysemous word that appears in both sentences.	Determine whether the word is used with the same sense in both sentences.	Accuracy.
WSC	Winograd Schema Challenge	Almost same as GLUE.		Accuracy.

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	34.81

Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
State of the art [9]	37.0
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	36.5
Oracle Rescoring of the Baseline 1000-best lists	~45

Table 2: Methods that use neural networks together with an SMT system on the WMT'14 English to French test set (ntst14).



Part 3

3주차진도 (2)

워드임베딩

워드 임베딩

| 희소 표현 (Sparse Representation)

원핫 인코딩, DTM 과 같이 벡터로 표현된 텍스트 데이터 대부분이 0으로 표현됨
 ⇒ 공간적 낭비 + 단어 간의 의미 표현 X

| 밀집 표현 (Dense Representation)

원래 단어의 집합 크기보다 작은 차원으로 0과 1만이 아닌 실수 값으로 표현
 => 벡터의 차원이 더욱 조밀해짐

원핫 인코딩, DTM으로 만든 행렬에 SVD를 적용하여 차원을 축소 시킬수 있긴 하지만, 어휘가 100만개 이상이라면 '100만 X 100만'이라는 행렬에 SVD를 적용하는 것조차 매우 많은 계산비용이 요구됨 ($O(n^3)$)



총 10,000개의 단어 코퍼스

고양이 = [0 0 0 0 0 1 0 0 0 ... 0 0 0]

원-핫 벡터 (희소 표현)

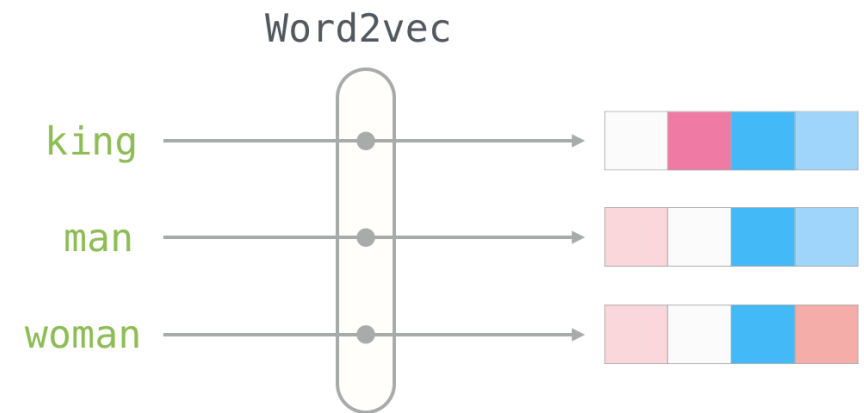
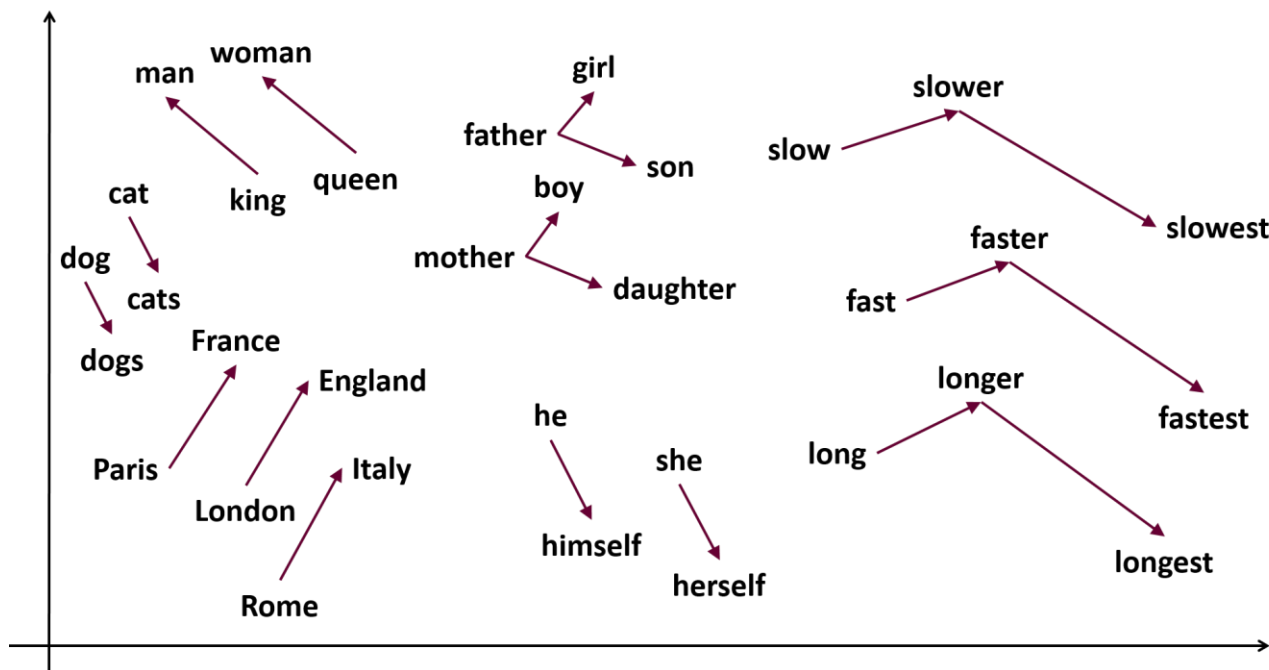
고양이 = [1.3 -2.7 0.9 ... 1.1 0.3]

밀집 표현

Word2Vec

I Word2Vec

: 인공 신경망을 활용하여 맥락 정보를 학습하고 단어의 분산 표현을 얻는다
 through 가중치 행렬 W



Word2Vec

I CBOW(Continuous Bag of Words)

: 주변에 있는 단어들을 입력으로 중간에 있는 단어들을 예측

"May the force be with you."

" " window size = 1 로 설정
" " 예측할 단어

0	0	1	0	0	0
---	---	---	---	---	---

각 단어들 원핫 벡터로 표시

0	1	0	0	0	0
---	---	---	---	---	---

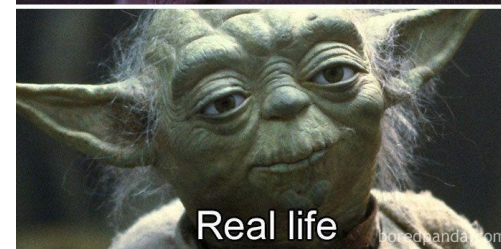
the

0	0	0	1	0	0
---	---	---	---	---	---

be

0.3	-1.3	2.2	4.1
1.1	8.06	4.0	3.2
6.3	7.8	7.77	2.0
0.02	0.8	2.8	4.5
0.27	8.7	0.00	-5.7
4.2	1.25	2.55	3.2

$W_{6 \times 4}$ W_{in}



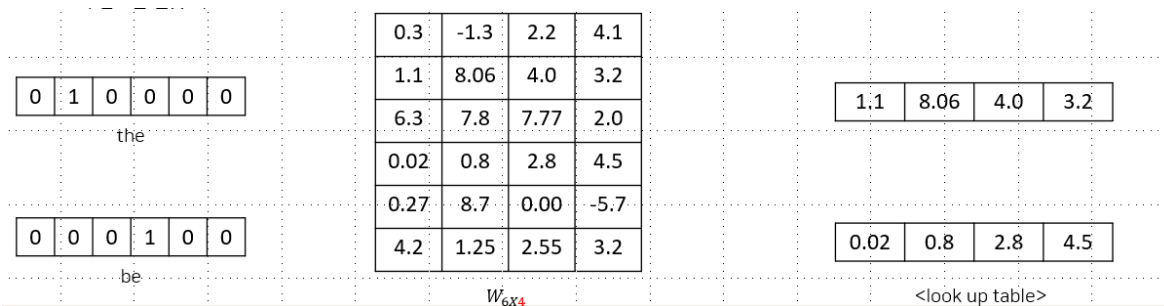
1.1	8.06	4.0	3.2
-----	------	-----	-----

0.02	0.8	2.8	4.5
------	-----	-----	-----

<look up table>

Word2Vec

CBOW(Continuous Bag of Words) "May the force be with you."



0.56	4.43	3.4	3.85
------	------	-----	------

$$V = \frac{V_{\text{the}} + V_{\text{be}}}{2 \times n (\text{window size})}$$

$W_{4 \times 6}$ W_{out}

0.45	4.5	5.69	1.85	4.4	9.36
------	-----	------	------	-----	------

Softmax

0.09	0.32	0.35	0.2	0.03	0.01
------	------	------	-----	------	------



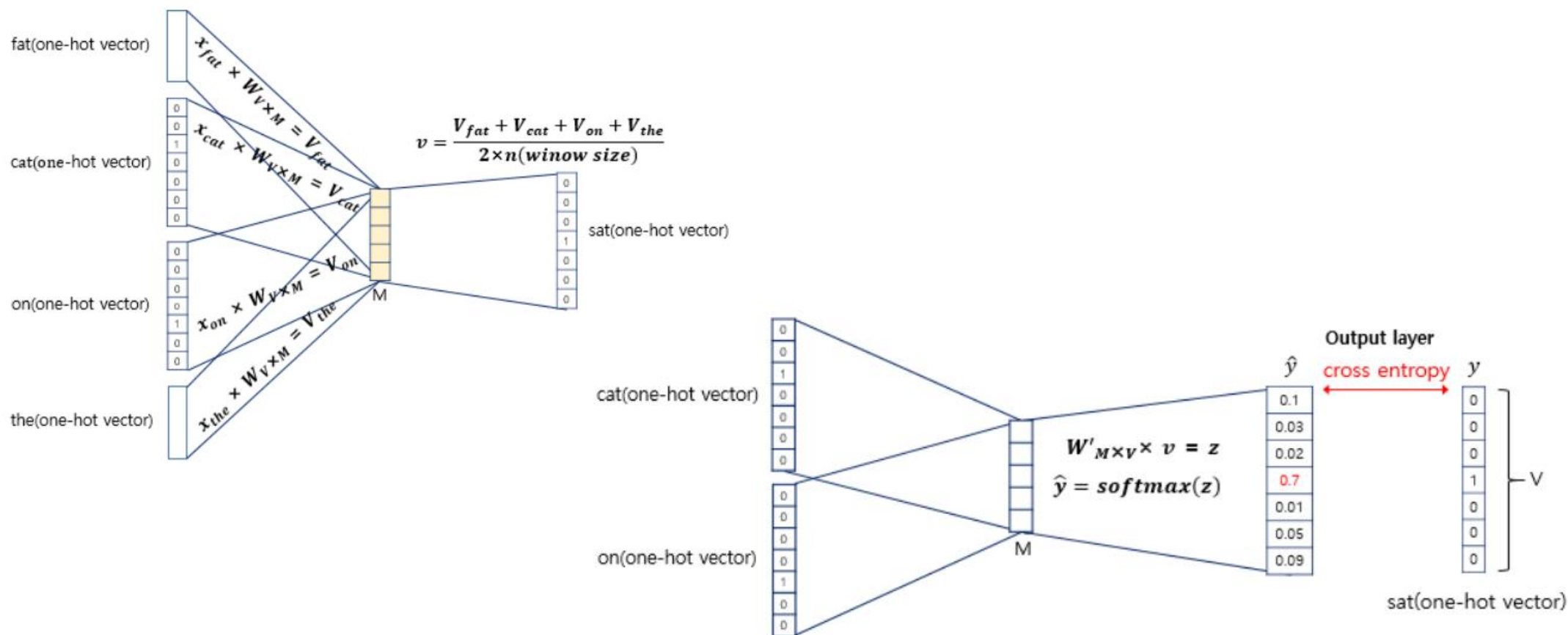
Loss Function (Cross Entropy)

0	0	1	0	0	0
---	---	---	---	---	---

정답 레이블

Word2Vec

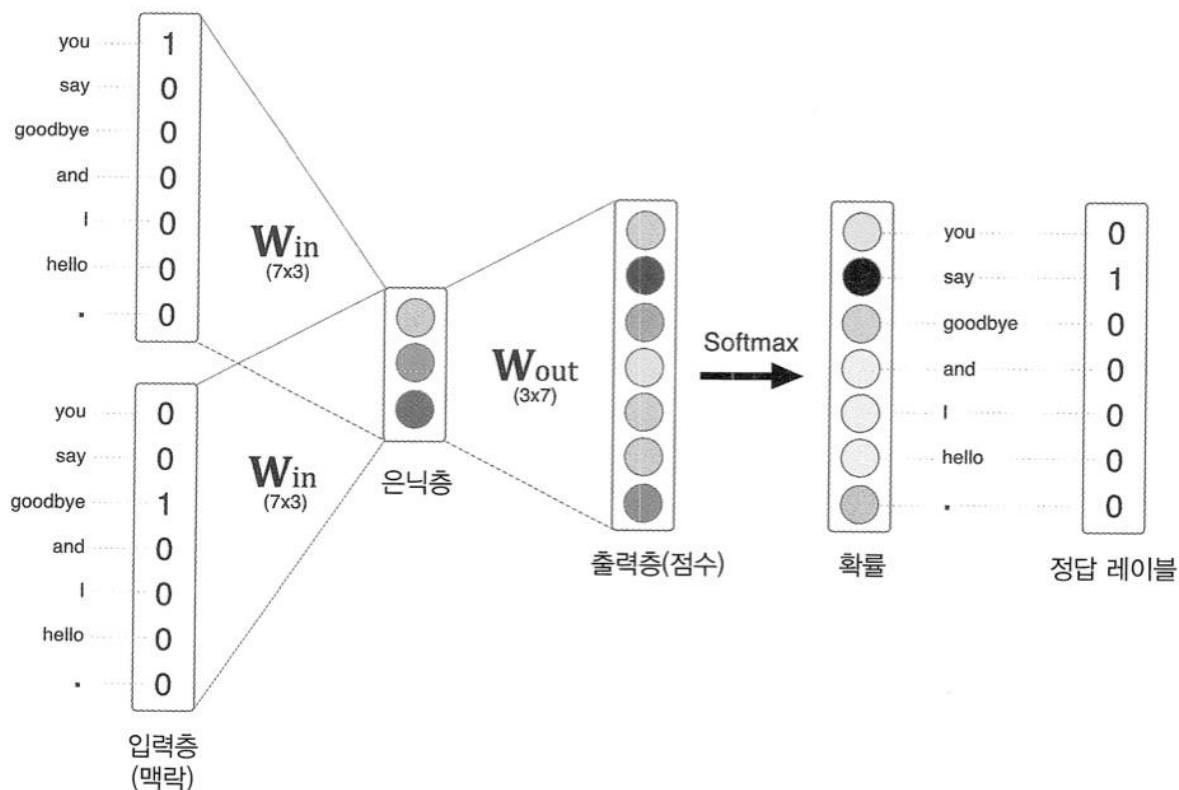
I CBOW(Continuous Bag of Words)



Word2Vec

I CBOW(Continuous Bag of Words)

그림 3-12 CBOW 모델의 구체적인 예(노드 값의 크기를 흑백의 진하기로 나타냄)



```
class SimpleCBOW:
    def __init__(self, vocab_size, hidden_size):
        V, H = vocab_size, hidden_size

        # 가중치 초기화
        W_in = 0.01 * np.random.randn(V, H).astype('f')
        W_out = 0.01 * np.random.randn(V, H).astype('f')

        # 계층 생성
        self.in_layer0 = MatMul(W_in)
        self.in_layer1 = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        self.loss_layer = SoftmaxWithLoss()

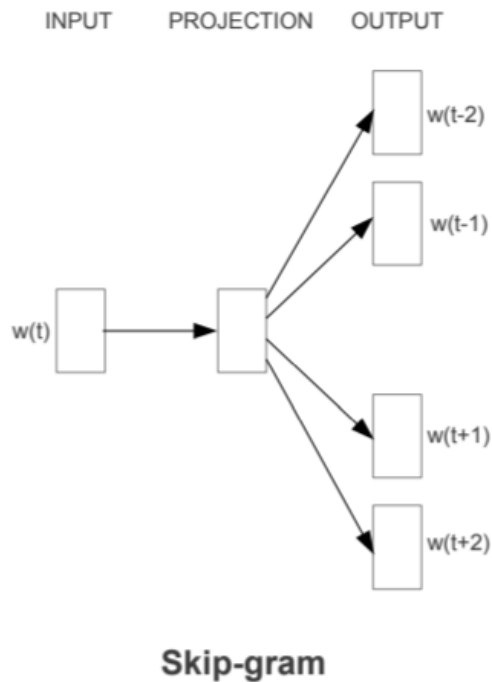
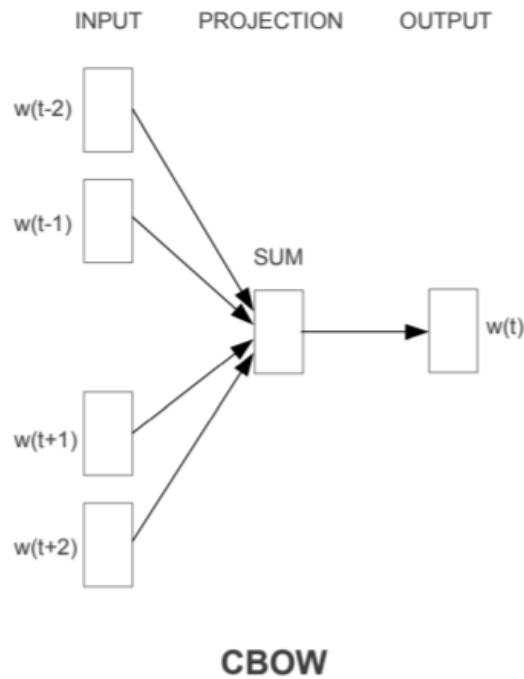
        # 모든 가중치와 기울기를 리스트에 모음
        layers = [self.in_layer0, self.in_layer1, self.out_layer]
        self.params, self.grads = [], []
        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads

        # 인스턴스 변수에 단어의 분산 표현을 저장한다.
        self.word_vecs = W_in
```


Word2Vec

I Skip-gram

: 중앙의 단어로부터 주변의 여러 단어(맥락)를 추측



Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

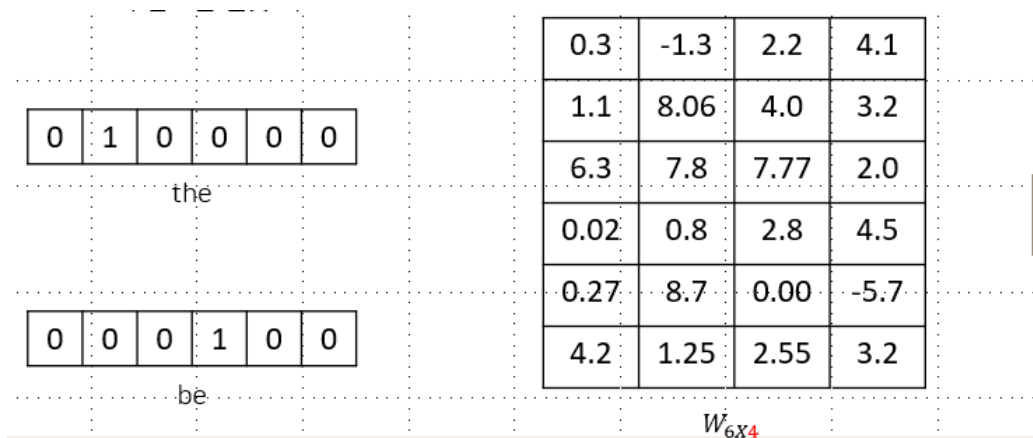
$$p(o|c) := \hat{y}_o = \frac{\exp(u_o^t v_c)}{\sum_{i=1}^K \exp(u_i^t v_c)}$$

c : 중심단어, o : 문맥

Word2Vec

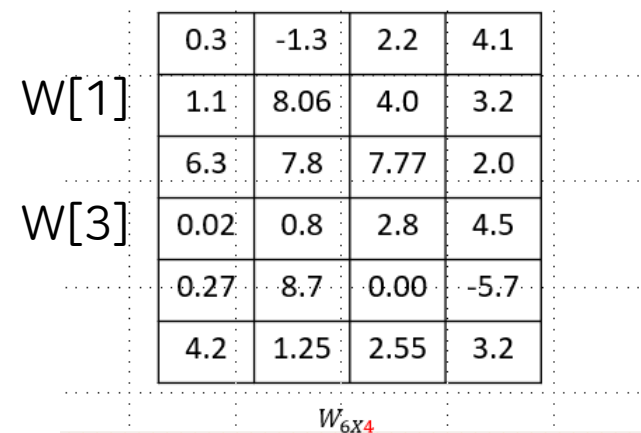
I 개선점 - Embedding 계층

: 신경망에 넣기 전 원핫벡터로 표현하는 부분에서 많은 계산 복잡도를 수반함
 => 단순히 단어 ID에 해당하는 행(벡터)을 추출하는 임베딩 계층을 만들어놓는다



- 단어에 정수 인코딩

{ "may": 1
 "the": 2
 "force": 3
 "be": 4
 "with": 5
 "you": 6 }



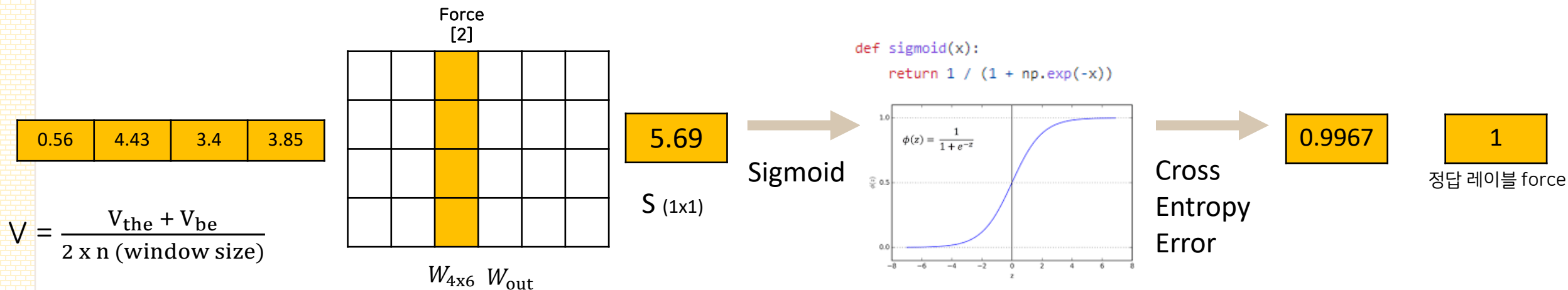
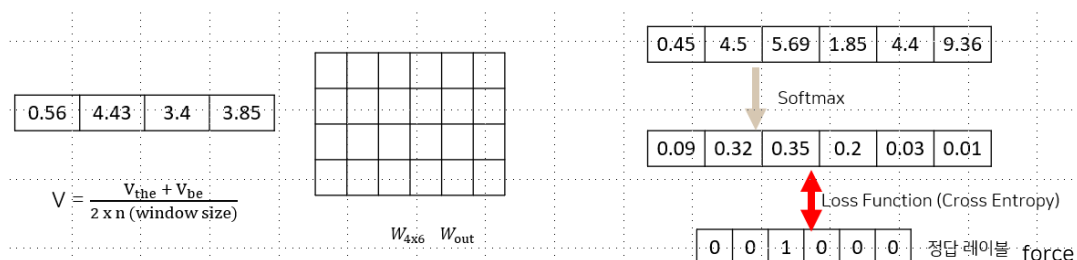
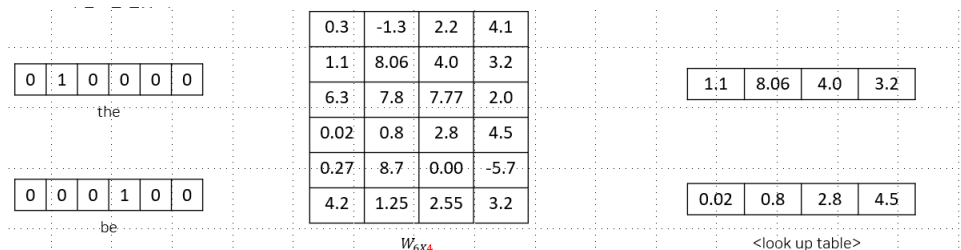
기존 입력층 부분을 단어 ID에 해당하는 행을 추출해오는 방식으로 변환
 ~ 메모리 사용량 줄이고 쓸데없는 계산도 생략 가능

Word2Vec

| 개선점 - Negative Sampling

: 다중 클래스 분류에서 이진 클래스 분류로 문제를 해결하여 연산량을 적게 한다

"May the force be with you."



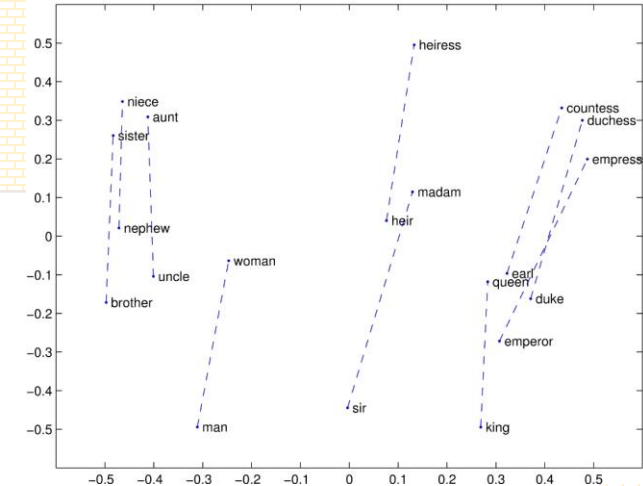
GloVe

| GloVe

: 카운트 기반 + 예측 기반 (실제값 - 예측값 오차 손실 함수 줄여 나가는)

단어 간 유사도X

사용자 지정 윈도우 내에서만 학습 진행. 코퍼스 전체의 동시발생(co-occurrence) 정보는 반영 X



동시 등장 확률과 크기 관계 비(ratio)	k=solid	k=gas	k=water	k=fasion
P(k ice)	0.00019	0.000066	0.003	0.000017
P(k steam)	0.000022	0.00078	0.0022	0.000018
P(k ice) / P(k steam)	8.9	0.085	1.36	0.96

중심 단어 k와 주변 단어 i의 내적이
전체 코퍼스 내에서의 동시 등장 확률이 되도록 임베딩 벡터를 학습

$$\text{dot product}(w_i, \tilde{w}_k) \approx \log P(k | i) = \log P_{ik}$$

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$F(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$w_i \longleftrightarrow \tilde{w}_k$$

$$X \longleftrightarrow X^T$$

$$F(X - Y) = \frac{F(X)}{F(Y)}$$

위 세가지 조건을 만족하는 함수 F는 지수함수!

$$J = \sum_{i,j=1}^V (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

FastText

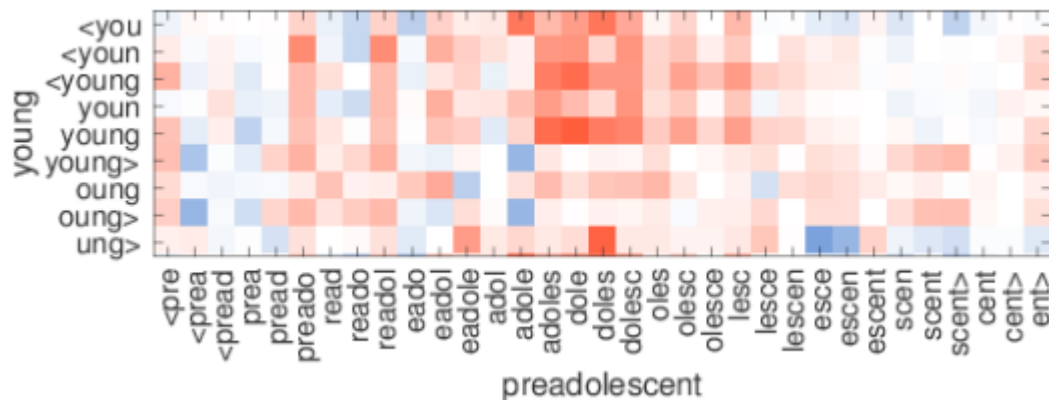
FastText

: n-gram 을 활용하여 형태소의 변화까지도 고려한 임베딩 벡터 구축

✓Word2Vec : 단어를 쪼개질 수 없는 단위로 생각

✓FastText : 하나의 단어 내에도 sub-word 개념으로 더 잘게 쪼갬

Subword model



◦ 모르는 단어 (OOV)에 대해서도 다른 단어와의 유사도 계산 가능

◦ 빈도 수가 적은 단어에 대해서 어떻게든 다른 단어들과 접점 만들 수 있음

| 단어를 n=3~6인 n-gram으로 나누고 각각을 다른 벡터에 할당



Part 4

예습과제리뷰

- 네이버 영화 리뷰 Word2Vec

```
targetXML = open('ted_en-20160408.xml', 'r', encoding='UTF8')
target_text = etree.parse(targetXML)

# xml 파일로부터 <content>와 </content> 사이의 내용만 가져온다.
parse_text = '\n'.join(target_text.xpath('//content/text()'))

# 정규 표현식의 sub 모듈을 통해 content 중간에 등장하는 (Audio), (Laughter) 등의 배경음 부분을 제거.
# 해당 코드는 괄호로 구성된 내용을 제거.
content_text = re.sub(r'#[^\n]*#', '', parse_text)

# 입력 코퍼스에 대해서 NLTK를 이용하여 문장 토큰화를 수행.
sent_text = sent_tokenize(content_text)

# 각 문장에 대해서 구두점을 제거하고, 대문자를 소문자로 변환.
normalized_text = []
for string in sent_text:
    tokens = re.sub(r"^[a-z0-9]+", " ", string.lower())
    normalized_text.append(tokens)

# 각 문장에 대해서 NLTK를 이용하여 단어 토큰화를 수행.
result = [word_tokenize(sentence) for sentence in normalized_text]
```

```
[ ] model = Word2Vec(sentences=result, size=100, window=5, min_count=5, workers=4, sg=0)
```

여기서 Word2Vec의 인자는 다음과 같습니다.

- size = 워드 벡터의 특징 값. 즉, 임베딩 된 벡터의 차원.
- window = 컨텍스트 윈도우 크기
- min_count = 단어 최소 빈도 수 제한 (빈도가 적은 단어들은 학습하지 않는다.)
- workers = 학습을 위한 프로세스 수
- sg = 0은 CBOW, 1은 Skip-gram.

XML 파일 전처리
데이터 내 필요없는 지시문 삭제 전처리
NLTK 토큰화

```
[ ] # 불용어 정의
stopwords = ['의', '가', '이', '은', '들', '는', '좀', '잘', '강', '과', '도', '를', '으로', '자', '에', '와', '한', '하다']
```

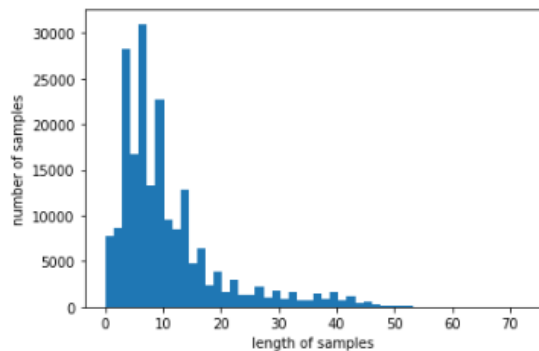
한국어 전처리

```
▶ # 형태소 분석기 OKT를 사용한 토큰화 작업 (다소 시간 소요)
okt = Okt()

tokenized_data = []
for sentence in tqdm(train_data['document']):
    tokenized_sentence = okt.morphs(sentence, stem=True) # 토큰화
    stopwords_removed_sentence = [word for word in tokenized_sentence if not word in stopwords] # 불용어 제거
    tokenized_data.append(stopwords_removed_sentence)
```

```
[ ] # 리뷰 길이 분포 확인
print('리뷰의 최대 길이 :', max(len(l) for l in tokenized_data))
print('리뷰의 평균 길이 :', sum(map(len, tokenized_data))/len(tokenized_data))
plt.hist([len(s) for s in tokenized_data], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

리뷰의 최대 길이 : 72
리뷰의 평균 길이 : 10.716703668146726



모델에 넣기 전에 간단하게라도
데이터의 개략적인 분포나 정보 파악하기

3. 사전 훈련된 Word2Vec

구글에서 제공하는 사전 훈련된 Word2Vec 모델을 가져올 것이다. 이 모델은 3백만 개의 단어를 300차원의 임베딩 벡터로 만들어놓은 것이다.

예시 코드를 사용하다 HTTP Error 404: Not Found 에러가 나는 경우는

<https://drive.google.com/file/d/0B7XkCwpl5KDYNINUTTISS21pQmM/edit>

위 링크에서 모델을 다운받고 압축을 푼 후 구글 드라이브 적당한 디렉토리에 넣어두고 실행하면 된다.

```
[ ] import gensim
    import urllib.request

# 모델 저장해놓은 경로를 적어주면 된다
word2vec_model = gensim.models.KeyedVectors.load_word2vec_format('/content/drive/MyDrive/kubig/GoogleNews-vectors-negative300.bin', binary=True)
```

민규 님

XML

XML (Extensible Markup Language) 데이터 파일이다. HTML 문서와 매우 유사한 형식이지만 사용자 정의 태그를 사용하여 각 객체 내에서 객체와 데이터를 정의한다. XML 파일은 텍스트 기반 데이터베이스로 생각할 수 있다.

XML 파일은 프로그램 간에 그리고 인터넷을 통해 데이터를 저장하고 전송하는 표준 방법이다.

```
[ ] targetXML = open('ted_en-20160408.xml', 'r', encoding='UTF8')
targetXML
```

```
<_io.TextIOWrapper name='ted_en-20160408.xml' mode='r' encoding='UTF8'>
```

etree

XML 파일을 객체화시켜준다.

구문분석 parsing

문장을 그것을 이루고 있는 구성 성분으로 분해하고 그들 사이의 위계 관계를 분석하여 문장의 구조를 결정하는 것을 말한다. 컴퓨터 과학에서 파싱은 일련의 문자열을 의미있는 토큰으로 분해하고 이들로 이루어진 파스 트리를 만드는 과정을 말한다.

```
▶ target_text = etree.parse(targetXML)
target_text
```

```
👤 <lxml.etree._ElementTree at 0x7feadb0edec0>
```

복습과제

- 코드 리뷰
(pytorch word embedding,
word2vec – fasttext)
- 논문 리뷰
(word2vec, GloVe, FastText)

다음주 진도

- <딥러닝을 이용한 자연어처리 입문>
Ch8. 순환 신경망
- <밑바닥부터 시작하는 딥러닝2>
Ch5. 순환 신경망(RNN)

예습과제

- 주가 예측 RNN



수고하셨습니다!

Contact

15기 김제성

✉ rlawptjd1409@korea.ac.kr

☎ 010-2609-5046