

## 언어학 학사 학위논문

# 한국어 형태적 특징을 반영한 단어 임베딩 모델 (Skipgram, Glove, Fasttext)의 통합적 고찰 —매개변수 조절과 감정분석에 적용한 성능비교—

### 초록

본 연구는 한국어의 특징을 고려해 Skipgram, Glove, Fasttext 3가지 모델에 대해 임베딩을 만들고 감정분석 과제를 통해 성능을 평가한다. 임베딩은 그 자체로만 쓰이는 경우는 거의 없고 항상 더 큰 딥러닝 모델과 결합해 쓰인다. 그러나 기존 연구에서 한국어 임베딩의 평가는 유추 혹은 유사성 문제를 통한 내재적 평가에 국한되어 있다. 본고는 이를 주목해 임베딩과 LSTM 과의 결합을 통해 감정분석으로써 성능을 평가한다. 본 연구는 실험을 통해 다음의 6가지를 중점적으로 확인하였다: (1) 단어보다 형태소가 한국어 임베딩 학습에 더 적합하다. (2) 위 3가지 모델 중 성능이 가장 좋은 모델은 음절을 n-gram의 입력 단위로 한 Fasttext이다. (3) Skipgram과 Glove의 가장 큰 차이는 전혀 등장하지 않는 단어  $l(w_i, c_j)$ 에 대한 비용 처리 방식이다. (4) 임베딩 차원의 크기 파라미터가 성능에 가장 영향을 많이 준다. (5) 최소출현 빈도와 최대 창크기의 크기 파라미터는 (단어, 문맥) 튜플 전체 집합  $D$ 의 분포에 영향을 준다. (6) 개별 모델의 특수성을 고려한 파라미터 최적화 방식이 존재한다.

주요어: Skipgram, Glove, Fasttext, Word2vec, 한국어, 단어 임베딩

2018 년 6 월

서울대학교 인문대학 언어학과

2012-10141

이한결

# 목차

목차.....	1
<b>1. 서론.....</b>	<b>3</b>
1.1. 연구의 배경.....	3
1.2. 연구의 목적.....	6
1.3. 논문의 구성.....	6
<b>2. 본론.....</b>	<b>8</b>
2.1. 이론적 배경: 임베딩 모델.....	8
2.1.1. 수리적 표기법.....	9
2.1.2. Word2vec.....	10
2.1.3. Fasttext.....	15
2.1.4. Glove.....	16
2.2. 실험소개.....	18
2.2.1. 연구과제.....	18
2.2.2. 실험 말뭉치.....	19
2.2.3. 임베딩 모델의 입력단위와 파라미터.....	20
2.3. 감정 분석.....	21
2.3.1. 감정 분석 네트워크 정의.....	21
2.3.2. 감정 분석 네트워크 훈련.....	23

2.4. 실험 결과 해석 .....	24
2.4.1. 실험 범주와 해석 방법 .....	25
2.4.2. 입력 단위에 따른 성능 비교 .....	27
2.4.3. 모델간 성능 비교 .....	29
2.4.4. 파라미터에 따른 성능 비교 .....	30
2.5. 새로운 파라미터 최적화 방법 .....	37
<b>3. 결론 .....</b>	<b>39</b>
3.1. 한계점 .....	39
3.2. 의의 .....	40
<b>참고문헌 .....</b>	<b>41</b>

# 1. 서론

## 1.1. 연구의 배경

분산가정은(Distributional Hypothesis) 유사한 문맥에서 등장하는 단어들이 서로 비슷한 의미를 가질 가능성이 높다는 것을 의미한다. Distributional vectors 또는 단어 임베딩은, 단어를 저차원의 실수 벡터로 표현하되 분산가정이 반영되도록 설계한다. 단어 임베딩이 분산 가정을 충실하게 반영하고 있다면 벡터간의 유사성을 재는 cosine similarity 등의 척도로 단어간의 유사성을 구할 수 있고 벡터간의 산술적 차이로 king:man=? : women과 같은 유추문제 또한 풀 수 있다.

특히 단어 벡터를 모은 단어 임베딩 행렬은, 품사 태깅(part of speech tagging), 개체명 인식(named entity recognition) 그리고 구문 분석(syntactic parsing)등을 처리하는 심층 신경망의 첫 번째 층을 구성한다. 단어 임베딩은 다양한 자연언어처리 과제의 획기적인 성능향상에 기여한 핵심 요인이다(Young, Hazarika, Poria, & Cambria, 2017).

단어 임베딩을 구하는 방법과 벡터들이 지니는 특성들은 Distributional Semantics의 영역에서 오랫동안 연구되어져 왔다(Baroni & Lenci, 2010; Turney & Pantel, 2010). 임베딩을 구하는 전통적인 방법들은 코퍼스의 공기 정보(co-occurrence statistics)를 활용해 도출했으며, 근래에 이르러서는 인공 신경망을 활용한 단어 임베딩 방법론들이 등장하였다(Bengio, Ducharme, Vincent, & Jauvin, 2003; Collobert & Weston, 2008). 인공 신경망이 가지는 계산상의 복잡도를 획기적으로 줄인 모델이 Mikolov et al.(2013a; 2013b)에 의해 공개되었으며, Word2vec은 단어 임베딩을 구하는 정석적인 방식으로 굳어졌다. 하지만 Mikolov et al. (2013a; 2013b)의 모델은 코퍼스 전체의 통계정보를 효과적으로 활용하지 못한다는 점이 Pennington, Socher, and Manning (2014)에서 언급되었다. 그들이 밝힌 Glove는 코퍼스 전체의

co-occurrence에 로그를 취한 행렬을 분해하는 방식이다. 한편, 앞선 방식들은 단어 보다 더 작은 단위의 정보를, 예를 들어 단어를 구성하고 있는 형태소의 정보를 학습에 활용하지 않는다. 근래에 등장한 FastText(Baroni & Lenci, 2010)에서는 Skipgram의 단어벡터를 n-gram의 Bag-of-Words로 나타냄으로써 단어의 형태적인 정보 또한 학습 가능하도록 개선하였다.

하지만 위 기법은 영어와 같은 고립어에 적용하기 적합하도록 개발된 임베딩 기법이다. 이 기법을 그대로 교착어인 한국어에 적용하기에는 무리가 있다. 한국어 코퍼스와 영어 코퍼스의 크기 대비 단어 등장 빈도수를 살펴보면, 영어는 단어당 빈도수가 일정 수 이상 유지되는 고른 분포를 보이지만, 한국어는 상당 부분의 단어가 10회 미만의 빈도로 집계된다. 특정 빈도수가 유지되지 않는 단어는 노이즈 정보로 작용해 다른 단어 벡터가 적절히 학습되는 것을 방해한다. 한국어와 같은 코퍼스에서 나타나는 단어당 빈도수간의 불균형은, 두 개 이상의 형태소가 결합에 어간을 만드는 교착어의 특성에서 비롯된다. 한국어에서는 “생각하고, 생각하니, 생각하며, 생각해서”와 같이 명사에 활용어미가 붙거나 “생각을, 생각은, 생각도, 생각조차”와 같이 조사가 붙어 단어를 만든다. 또, 한국어의 상당부분의 단어는 한자에 기반한다. 같은 한자를 공유하는 단어는 의미적 유사성을 띠기도 한다.

한국어에서의 임베딩 기법 적용은 단어 보다 더 작은 단위의 정보들이 학습에 이용되게끔 하는 방식으로 초점이 맞추어져 있다. 유홍연와 고영중 (2017)에서는 Bidirectional LSTM을 사용해 음절 단위에서 단어벡터를 합성하는 방식을 택하였고, Kim, Jernite, Sontag, and Rush (2016)와 최상혁 (2017)에서는 컨볼루션 레이어를 활용해 글자 단위의 임베딩을 합성해 단어벡터를 만든다.<sup>1</sup> Ceyda and Byoung-Tak (2016)에서는 RNN계열의 언어모델로 도

---

<sup>1</sup> Kim et al. (2016)과 Choi et al.(2016)에서는 글자 임베딩 레이어에 다양한 필터를 적용해 피쳐맵을 만들고, 이후 최대 풀링 레이어(Max Pooling)를 거쳐 단어 벡터를 산출한다. 그런데 Choi et al.(2016)에서 Skipgram의 negative sampling 방식으로 단어 벡터를 학습하는 반면 Kim et al.(2016)에서는 Highway Network와 LSTM으로 언어모델을 구축해 단어 벡터를 학습한다.

출된 임베딩이 Skipgram방식의 단어 임베딩보다 통사적 정보를 더 잘 학습한다는 점에 주목하였다. 그들은 Fasttext모델의 단어 임베딩과 C2W(Ling et al., 2015) 모델을 결합해 단어 임베딩을 구한다.<sup>2</sup> RNN 혹은 CNN을 통해 음절 혹은 문자소 단위에서부터 단어 벡터를 합성해가는 방식은, 형태소 분석기를 필요로 하지 않는다는 점에서 이론적 장점이 있다. 하지만 실제로 비슷한 형태소를 공유하고 있는 단어가 비슷한 공간에 맵핑되기 위해서는 각 형태소를 공유하고 있는 단어 간의 문맥 또한 유사해야 한다. 이는 단어의 연쇄를 통해 다음 단어를 예측하는 언어 모델의 학습 방식에서 오는 특징이다. 단어를 입력으로 한 RNN 혹은 CNN 언어모델은 코퍼스의 크기와 단어 분포에 성능이 크게 좌우된다. 때문에 한국어 코퍼스의 특징적인 단어 빈도 불균형 문제를 완전히 해결했다고 보기 힘들다. 또 Skipgram, Cbow, Glove, Fasttext과 같이 깊이가 얇은 모델에 비해 학습에 더 많은 시간이 소요되고 임베딩 레이어를 단일 층으로 구성할 수 없다는 단점이 있다. RNN과 CNN으로 학습한 임베딩을 다른 딥러닝과 결합하기 위해서는 벡터 합성 과정에 동원되었던 레이어를 전부를 가져와 임베딩 레이어로 구성해야 한다.

한편, 앞서 소개한 모델들(Skipgram, Cbow, Glove, Fasttext)를 그대로 사용하되 단어가 아닌 형태소 임베딩을 구하는 방식을 택함으로써 한국어에서 보이는 단어 빈도의 불균형 문제를 우회하기도 한다. 최상혁, 설진석, 그리고 이상구 (2016)에서는 형식형태소를 제거한 실질적 의미가 뚜렷한 체언과 용언만을 입력으로 해 Skipgram, Cbow 그리고 Glove에 대해 실험하였다. 조현수와 이상구 (2017)에서는 Fasttext의 입력단위로 음절단위의 n-gram 형태소를 선택한다.

RNN과 CNN계열의 언어모델이 단어 빈도 불균형 문제로부터 자유롭지 못하며 학습 속도와 임베딩 레이어 결합의 관점에서 기존의 모델에 비해 큰 장점이 없다고 판단하여, 본 논문에서는 형태소를 입력단위로 한 Skipgram, Cbow, Glove, Fasttext를 기본 모델로

<sup>2</sup> Cinarel and Zhang (2016)에서는 Fasttext와 C2W의 결합방식으로 연결방식과 게이트 선택 방식을 이용한다.  $w = [w^{word}; w^{char}]$  또는  $w = (1 - g_w)w^{word} + g_w w^{char}$ ,  $g_w = \sigma(v_g^T x_w^{word} + b_g)$ .  $w^{word}$ 와  $w^{char}$ 는 각각 Fasttext와 C2W모델에서 도출된 단어벡터를 의미한다.

사용하였다.

## 1.2. 연구의 목적

Schnabel, Labutov, Mimno, and Joachims (2015)에서는 임베딩의 질을 측정하는 방법을 크게 내재적 방법과 외재적 방법 두 가지로 분류한다. 내재적 방법은 단어간의 통사적, 의미적 정보에 대한 평가를 단어 임베딩 자체만으로 측정하는 반면, 외재적인 방법에서는 단어 임베딩을 다른 모델의 입력 피쳐로 사용해 감정분석, 개체명 인식, 품사 분석등의 주요 자연언어처리 과제의 성능 향상에 기여하는지를 측정한다.

한국어에서 임베딩 성능에 관한 기존 연구는 대부분이 유사도 또는 관련도를 측정한 내재적 방법에 국한되어 있다(최상혁, 2017; 최상혁 et al., 2016). 그런데 단어 임베딩이 독립적으로 쓰이는 경우는 거의 없고, 항상 더 큰 딥러닝 모델과 결합되어 쓰인다는 현실에 비추어 본다면, 내재적 평가보다는 외재적 평가가 더 합리적이다. 현재 한국어 임베딩에 대한 외재적 평가에 대한 연구는 전무하다. 따라서 본 연구에서는 앞서 언급한 Word2vec, Glove 그리고 Fasttext 모델을 활용해 임베딩을 만들고 감정분석을 통해 임베딩의 질을 외재적 방법으로 평가하고 측정된 결과를 바탕으로, 임베딩 기법간에 차이와 임베딩의 질에 영향을 주는 핵심적인 파라미터가 무엇인지 알아보려고 한다. 나아가 각 모델의 특수성을 살린 새로운 파라미터 최적화 방법을 제안하고자 한다.

## 1.3. 논문의 구성

본 논문의 2.1에서는 실험해 사용한 임베딩 모델들의 개념을 구체적으로 정리하였다. 해당 절에서는 임베딩 모델의 파라미터에 대한 설명과 각 모델간의 핵심적 차이를 기술하였다. 2.2는 실험 진행에 관한 내용으로 실험에 사용한 말뭉치, 임베딩 모델 그리고 파

라미터 조합에 관한 간략한 소개가 포함된다. 2.3은 임베딩 질을 평가하기 위한 단계로, 학습을 위한 신경망 정의와 데이터 처리 과정을 기술한다. 2.4는 실험 결과를 바탕으로 임베딩 모델간의 차이와 모델 내 파라미터간의 영향에 대해 기술한다. 2.5는 2.4를 기반으로 새로운 파라미터 최적화 방법을 제안한다. 마지막 3장에서는 결론과 본 연구의 한계점에 대해 서술한다.



## 2. 본론

### 2.1. 이론적 배경: 임베딩 모델

본 논문에서 사용한 Cbow, Skipgram, Fasttext, Glove는 인공 신경망을 사용하는 예측기반 방식(prediction based methods)과 행렬 분해를 이용하는 통계기반 방식(count based methods)으로 나눌 수 있다. Cbow, Skipgram, Fasttext는 예측기반 방식으로 분류된다. 이 방식은 문장 안에 일부 단어를 보고 나머지 단어를 맞추는 방식으로 학습한다. 예를 들어 “백설 공주는 빨간 \_\_\_ 먹는다”라는 문장에서 인공 신경망은 “사과”를 맞추므로써 단어에 대한 의미를 인공 신경망 안에 적절히 쌓아 나간다. 학습이 끝난 후 인공신경망의 웨이트를 임베딩 행렬로 사용한다.

한편 Glove와 같은 통계 기반 방식은 학습 단어가 다른 단어와 얼마나 자주 등장했는가에 대한 통계 정보를 활용한다. 가령 (백설, 공주) 혹은 (빨간, 사과)와 같은 단어 쌍들의 빈도를 통계로 단어-문맥 공기 행렬을 만들고 이를 분해하여 임베딩 행렬을 만든다. 단어 벡터가 적절히 학습되었다면, 임베딩 행렬로부터 원래의 단어-문맥 공기 행렬을 큰 차이 없이 복원할 수 있다.

본 연구에 사용한 모델들은 구체적인 학습 목표와 방식에서 서로 다르나, 코퍼스를 단어-문맥의 관점에서 보고 학습 데이터를 만든다는 점에서는 동일해 국소 문맥 창(local window context) 기법으로 분류될 수 있다. 즉 본 연구에서 소개한 모델은 모두 특정 단어를 기준으로 좌우에 일정 크기의 창을 두고 문맥을 정의해 학습 데이터를 가공한다. 가령 창의 크기가 1일 경우 “백설 공주는 빨간 사과를 먹는다”라는 문장을 “[(백설, 공주는), (공주는, 빨간), (빨간, 사과를), (사과를, 먹는다)]”와 같이 가공한다. 이후 예측 기반 방식은 (단어, 문맥)에서 단어를 보고 문맥을 맞추거나 문맥을 보고 단어를 맞춘다. 통계기반 방

식은 (단어, 문맥) 튜플로부터 단어-문맥 공기 행렬을 만든다.

각 임베딩 모델의 세부적인 사항은 뒤따르는 절에서 소개한다. 2.1.1에서는 공통된 수리적 표기법을 정리한다. 2.1.2와 2.1.3에서는 예측 기반 방식의 Word2vec(Cbow, Skipgram), Fasttext를 2.1.4 에서는 통계기반 방식의 Glove를 소개한다.

### 2.1.1. 수리적 표기법

임베딩 모델은 코퍼스에 등장하는 각 단어에 대해 적절한 벡터를 찾는 것을 목표로 한다. 학습의 대상이 되는 단어벡터는 그 단어와 같이 등장하는 문맥 단어벡터와의 특수한 관계를 만족시켜야 한다. 임베딩 모델마다 중앙 단어-문맥 벡터에 대한 특수한 관계식이 정의되고 학습방식에도 차이가 난다. 아래 표는 임베딩 모델간의 수리적 모델을 기술하기 앞서 공통된 표기를 정리한 것이다. 표기는 Levy, Goldberg, and Dagan (2015)의 것을 따른다.

임베딩 학습에 앞서 원시 코퍼스를 특수문자와 기호를 제외해 단어만 남도록 한 후, 코퍼스를 단어 단위로 잘라 나타낸 리스트를  $K(=[k_1, k_2, \dots, k_N])$ 로 정의한다. 또 각 단어  $k_t$  마다 특정 크기의 창을 두어 문맥을 설정할 수 있다. 가령 창의 크기가  $s$  라면  $k_t$  의 문맥은  $\{k_i | t-s \leq i \leq t+s, i \neq t\}$ 가 된다.

코퍼스에 등장하는 단어를 중앙 단어  $w \in V_w$ 와 문맥 단어  $c \in V_c$ 로 나누어 지칭한다. 또 코퍼스 상에서 관찰된 단어-문맥 쌍은  $(w, c) \in D$ 로 정의한다. 이 단어-문맥 쌍이 나타나는 빈도는  $\#(w, c)$ 로 표기하고, 이때 코퍼스와 단어-문맥 쌍 간에는 다음의 관계식을 만족시킨다.  $\#(W) = \sum_{c \in V_c} \#(w, c)$ ,  $\#(C) = \sum_{w \in V_w} \#(w, c)$ .

각 중앙 단어  $w \in V_w$ 와 문맥 단어  $c \in V_c$ 에 대응되는  $d$ 차원의 벡터를 각각  $\vec{w}$ ,  $\vec{c} \in \mathbb{R}^d$ 로 정의한다.  $\vec{w} \in W$ 일 때 집합  $W$ 는 중앙단어  $w$ 에 대한 크기  $|V_w| \times d$  행렬로 표현 가능하다. 어떤 단어 벡터  $\vec{w}$ ,  $\vec{c}$ 에 대해  $W$ 와  $C$ 의 특정 행으로 쉽게 지칭할 수 있

는 장점이 있기 때문에 본고에서는  $W$ 와  $C$  각각을 단어와 문맥에 대한 임베딩 행렬의 의미로 쓴다.

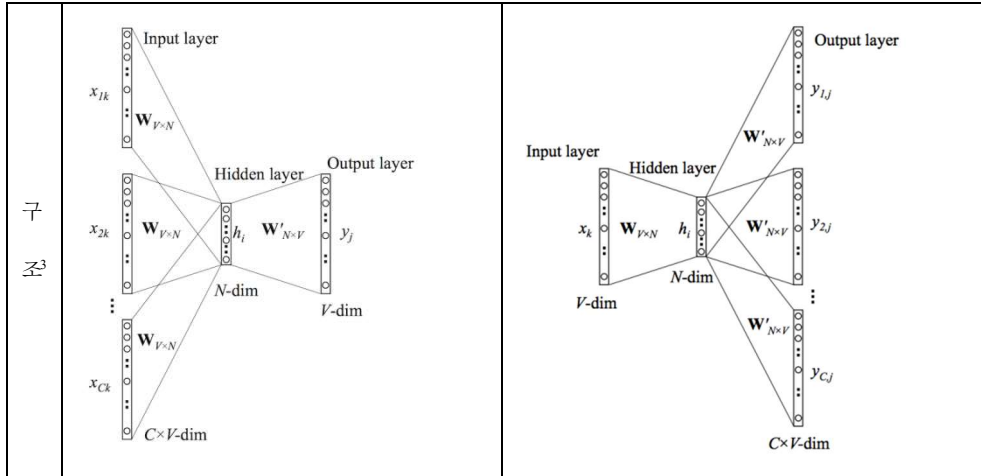
표기	의미
$V_w, V_c$ $W, C$ $D$ $K$	$V_w, V_c$ : 단어 전체 집합 $W, C$ : 임베딩 행렬 $D$ : (단어-문맥) 튜플 전체 집합 $K$ : 단어 리스트
$w \in V_w, c \in V_c$ $(w, c) \in D$ /	$w, c$ : 단어 $(w, c)$ : (단어-문맥) 튜플
$\vec{w}, \vec{c} \in R^d$ $\vec{w} \in W, \vec{c} \in C$ $W_c$	$\vec{w}, \vec{c}$ : $d$ 차원의 단어 벡터 $W$ 에서 단어 $c$ 에 대한 행 벡터
$\#(W) = \sum_{c \in V_c} \#(w, c),$ $\#(C) = \sum_{w \in V_w} \#(w, c)$	$\#$ : 빈도수

## 2.1.2. Word2vec

Word2vec은 Mikolov et al.(2013a; 2013b)에서 공개된 단어 임베딩 툴킷으로 전처리 스크립트와 Skipgram, Cbow 두 모델을 포함한다. 두 모델 모두 신경망의 출력으로 단어에 대한 확률을 내보낸다는 점에서 예측기반 임베딩 모델로 분류될 수 있다. Cbow 모델의 경우 학습 단어를 보고 문맥 단어의 확률을 예상하고, Skipgram은 문맥을 보고 중앙 단어에 대한 확률을 예상한다. Cbow 모델의 학습목표는  $\frac{1}{|K|} \sum_{t=1}^{|K|} \log p_{\theta}(v_t | v_{t-\frac{s}{2}} \dots v_{t+\frac{s}{2}})$  같이

정의될 수 있으며, Skipgram 모델은  $\frac{1}{|K|} \sum_{t=1}^{|K|} \sum_{j=t-s}^{t+s} \log p_{\theta}(v_j|v_t)$  로 정의된다. Cbow에서의  $v_{t-\frac{s}{2}} \dots v_{t+\frac{s}{2}}$  와 Skipgram에서의  $v_j$  는 학습 단어에 대한 문맥을 의미한다.  $p_{\theta}(\dots)$ 의 역할을 인공신경망이 담당하게 된다. 인공신경망은 입력, 은닉, 출력 3개의 층으로 구성되어 있으며, Cbow 모델의 경우 1-hot 벡터로 표현된 단어들을 입력으로 받아, 문맥 단어들에 대한 벡터를 중앙 단어들에 대한 확률을 출력으로 내보낸다. 반대로 Skipgram 모델은 중앙 단어를 입력으로 받아, 문맥 단어에 대한 확률을 계산한다. 두 모델 모두 입력층과 출력층의 노드의 개수는 코퍼스에 등장하는 단어의 수  $\#set(V_w)$ 으로 동일하다. 학습이 끝난 후 신경망의 가중치( $\theta$ ) 가운데 입력층과 은닉층을 잇는  $W$ 를 임베딩 행렬로 사용한다.

모델	Cbow	Skipgram
학습목표	$\frac{1}{ K } \sum_{t=1}^{ K } \log p_{\theta}(v_t v_{t-\frac{s}{2}} \dots v_{t+\frac{s}{2}})$	$\frac{1}{ K } \sum_{t=1}^{ K } \sum_{j=t-s}^{t+s} \log p_{\theta}(v_j v_t)$
은닉층	$h = \frac{1}{S} W^T \left( \sum_{i=1}^t x_i \right)$	$h = W^T x_t$



**Table 1 Word2vec 모델 설명**

출력층의 각 노드에 대한 확률 함수는 softmax를 통해 정의된다. 가령 Skipgram의 단어-문맥쌍의 softmax계산은  $p(w|k) = \frac{\exp(h \times c_k^T)}{\sum_{j \in V_W} \exp(h \times c_j^T)}$  이다. softmax 함수는 모든 노드에 대해서 출력값을 계산해야 하기에 코퍼스의 크기가 커질 때, 분모의 연산량이 문제가 된다.

이 문제를 Mikolov et.al (2013a)에서는 hierarchical softmax를 이용해 해결하였고,<sup>4</sup> 이후 Mikolov et.al (2014b)에서는 negative sampling 방식을 이용하였다. Negative sampling 방식은 하나의 단어-문맥쌍 (w, c) 에대한 손실을  $E = -\log \theta(\vec{w} \times \vec{c}) - k \cdot X_{c_N \sim P_D} [\log - \vec{w} \times \vec{c}_N]$

<sup>3</sup> Cbow and Skipgram By Moucrowap [CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>)], from Wikimedia Commons

<sup>4</sup> Mikolov et.al (2013a)에서는 계산량의 부담을 줄이기 위해 hierarchical softmax를 소개하고 있다. 이 방식은 모든 노드에 대해 값을 계산하는 대신, 허프만 트리를 구성해 일부 노드에 대해서만 확률을 계산한다. 단어의 빈도수를 고려해 트리를 만들면 자주 등장하는 단어는 뿌리 노드와 가깝게 위치하게 된다. 단어-문맥쌍에 대한 계산량은 허프만 트리의 뿌리 노드에서 잎사귀 노드까지의 경로에 비례한다. 점근적 분석에 따르면 기존 계산량은  $|V|$ 에서  $\log |V|$ 로 줄어든다.

로 정의한다.<sup>5</sup> 나타나지 않는 단어는 분포  $P_D(c) = \frac{\#c^4}{Z}$  를 정의해 일부만 샘플링해 계산한다.

목표함수는 (단어, 문맥) 튜플 전체 집합  $D$  에 대해 정의되어 있지만 샘플을 특정 batch 크기로 나누고, SGD(stochastic gradient descent)를 이용해 차례대로 최적화한다. 원 논문의 코드에서는 batch 크기가 1인 비동기적(Asynchronous) SGD를 활용해 학습속도를 높였다.

(단어, 문맥) 튜플 전체 집합  $D$  의 분포는 최대 창 크기(maximum window size)<sup>6</sup>, 최소출현빈도(minimum occurrence) 그리고 서브 샘플링 비율(subsampling rate) 파라미터에 영향을 받는다. 창 크기에 따라 특정 단어-문맥쌍이  $D$  에 포함되기도 하고  $\#(w,c)$  의 빈도수를 자체를 바꾸기도 한다. Table2는 “정의를 세우는 콜트, 콜텍 노동자들 이야기”를 예문으로 해 창 크기가 바뀔 때  $D$  가 바뀌는 상황을 나타낸 것이다.

최소 출현 빈도는 특정 출현 빈도 미만의 단어를 코퍼스 전체에서 모두 삭제시켜 나머지 단어의 학습을 돕는 역할을 한다. 문맥창을 정의하기 이전에 파라미터가 적용되고, 단어 문맥 창 크기의 실질적 크기를 증가시키기도 한다. 이는 최소 출현빈도 파라미터를 적용하기 이전에는 포함되지 않았던 단어들이 일부 학습 단어 문맥에 포함되기 때문이다.

서브 샘플링 비율 파라미터( $t$ )도 문맥창을 만들기 이전에 적용된다. 단어마다 특정 확률  $p$ 로 샘플링 함으로써 자주 등장하는 단어에 대한 학습 횟수를 줄인다. 비율  $t$ 와 각 단어별 빈도  $f$ 로 확률  $p = 1 - \sqrt[t]{f}$  를 구한다. 최소 출현 빈도 파라미터와 마찬가지로 문

<sup>5</sup> 본 연구에서는 negative sample의 수를 Mikolov et al. (2013b)에서 제안하는 값 5로 정한다( $k=5$ ).

<sup>6</sup> Word2vec에서는 중심 단어에서부터 떨어진 거리에 따라 단어-문맥쌍의 가중치를 달리하는 방법을 택하고 있다. 만약 최대 창 크기가 5이면 중심 단어로부터 떨어진 거리에 따라  $\frac{5}{5}, \frac{4}{5}, \frac{3}{5}, \frac{2}{5}, \frac{1}{5}$ 의 가중치를 가진다. 이 가중치를 부여하는 방법으로 Word2vec에서는 각 단어마다 1부터 5 사이의 숫자를 무작위로 뽑아 창 크기를 설정한다.

맥창의 실질적 크기를 증가 시킬 수 있다.

본 연구에서는 서브 샘플링 비율을  $t = 10^{-5}$ 로 고정시키고 최소 출현 빈도를 10, 20, 50, 100으로 변화시킴에 따라 성능차이를 보고자 한다. 실험에 사용된 코퍼스가 크지 않아 정지 단어(stop word)에 대한 빈도를 줄이는 것보다 빈도수가 적은 단어에 대한 제거가 성능에 더 큰 영향을 미칠 것으로 기대되기 때문이다.

Sliding Window (Size=1)	Target Word	Context Word
[정의를 세우는]	정의를	세우는
[정의를 세우는 쿨트]	세우는	정의를, 쿨트
[세우는 쿨트, 쿨텍]	쿨트	세우는, 쿨텍
[쿨트, 쿨텍, 노동자들]	쿨텍	쿨트, 노동자들
[쿨텍, 노동자들, 이야기]	노동자들	쿨텍, 이야기
[노동자들, 이야기]	이야기	노동자들
Sliding Window(Size = 2)	Target Word	Context Word
[정의를 세우는, 쿨트]	정의를	세우는, 쿨트
[정의를 세우는 쿨트, 쿨텍]	세우는	정의를, 쿨트, 쿨텍
[정의를, 세우는 쿨트, 쿨텍, 노동자들]	쿨트	정의를, 세우는, 쿨텍, 노동자들
[세우는, 쿨트, 쿨텍, 노동자들, 이야기]	쿨텍	세우는, 쿨트, 노동자들, 이야기
[쿨트, 쿨텍, 노동자들, 이야기]	노동자들	쿨트, 쿨텍, 이야기
[쿨텍, 노동자들, 이야기]	이야기	쿨텍, 노동자들

Table 2 예시 “정의를 세우는 쿨트, 쿨텍 노동자들 이야기”

### 2.1.3. Fasttext

단어 단위를 입력으로 한 Word2vec 모델은 같은 형태소를 공유하고 있는 단어일지라도 동떨어진 벡터 공간에 맵핑될 수 있다는 단점이 있다. 이는 단어들 사이에서 파라미터를 공유하지 않기 때문에 생기는 문제이다. Fasttext는 단어를 n-gram의 BoWs(Bag of Words)로 표현함으로써 위 문제를 해결한다. 단어를 더 작은 단위의 벡터의 합으로 표현하면, 같은 gram을 공유하고 있는 단어가 같이 학습된다. Table3의 “소프트웨어” 예시는 음절단위 3-gram을 나타낸 것이다. 본 논문에서는 최소 n-gram으로 n=1을 사용하였다.

Fasttext는 Word2vec의 Skipgram 모델을 그대로 가져다 쓰되, 단어 벡터 부분을 n-gram 벡터의 합으로 바꾸어 표현한다. 유사도 함수  $s(c, w)$  만을 Table3과 같이 바꾼다. 본 논문에서는 음절과 문자소(자음/모음)을 Fasttext 모델의 gram 단위로 사용한다. 자모 수준의 gram에서는 하나의 자음 혹은 모음으로 이루어진 형태소에 대한 학습도 가능할 것으로 기대된다.

	Fasttext	Skipgram
출력층의 softmax 계산	$p(c w) = \frac{e^{s(c w)}}{\sum_{j=1}^{V_W} e^{s(c w)}}$	
$s(c, w)$	$s(c, w) = \sum_{g \in g_w} z_g^T C_c$	$s(c, w) = W_w * C_c$
Ex) 소프트웨어 <sup>7</sup> , n=3	$g_w \text{ 소프트웨어} = [ < \text{소프}, \text{소프트}, \text{프트웨}, \text{트웨어}, \text{웨어} >, < \text{소프트웨어} > ]$	$W^w \text{ 소프트웨어} = < \text{소프트웨어} >$

<sup>7</sup> N-gram에 대한 표기는 Jo and Lee(2017)의 것을 참고했다. 단어 앞에 시작과 끝 기호인 ‘<’, ‘>’를 포함해 BoW를 구성하되, 구분자를 포함한 전체 단어도 Bow에 포함된다.



Table 3 Fasttext와 Skipgram의 차이점

## 2.1.4. Glove

Skipgram 모델은  $D$ 의 단어-문맥쌍을 차례로 훑으면서 인공신경망의 단어 예측 확률을 높여가는 온라인 학습을 사용한다. 이 방식은  $D$  전체의 통계를 효과적으로 이용할 수 없다는 단점이 있다. Glove는 Skipgram과는 달리 코퍼스 전체 통계정보를 직접적으로 이용한다. Glove에서는 Skipgram과 마찬가지로 단어마다 문맥창을 설정해  $D$ 를 모은다. 하지만  $D$ 로부터 코퍼스의 단어-문맥 공기 행렬(co-occurrence)  $X_{i,j}$ 를 만들어 학습에 이용한다는 점에서 다르다. Glove에서는  $X_{i,j}$ 를 만든 후, 행렬 분해 알고리즘을<sup>8</sup> 통해 임베딩 행렬  $W$ 를  $\log(X_{i,j})$ 로부터 도출한다.

한편, 학습 측면의 관점에서는 Skipgram과 Glove가 완전히 다른 임베딩 기법으로 보이지만 두 기법간에는 상당한 유사한 면이 있다. 먼저 Glove의 목표함수에서 등장하는  $f(X_{i,j})$ 는 Word2vec의 subsampling 파라미터와 유사한 역할을 한다.  $f(X_{i,j})$ 는 자주 등장하는 단어에는 가중치를 적게 주고, 반대로 너무 적게 등장하는 단어의 경우 가중치를 더 주어 빈도수를 보정하는 역할을 한다. 또  $\log(X_{i,j}) = -\infty$ 인 경우를 고려해 전혀 등장하지 않는 단어( $X_{i,j} = 0$  일 때) 문맥에 대한 비용  $l_G(w_i, c_j)$ 를 0으로 만든다.

문맥창의 위치에 따라 가중치가 달리 부여된다는 점도 동일하다. Skipgram에서는 중심 단어로부터의 거리  $d$ 와 문맥창의 크기  $L$ 로 가중치  $w_{\text{Skipgram}} = \frac{L-d+1}{L}$ 로 산출하고

---

<sup>8</sup> 행렬 분해(matrix factorization)를 통한 임베딩 기법은 차원이 높은 행렬을 차원이 낮은 임베딩 행렬들로 나타내는 것을 의미한다. Glove에서는 로그를 취한 단어-문맥 공기 행렬  $X_{i,j}$ 를 학습 단어, 문맥 임베딩 행렬  $W, C$ 로 나타내는 것을 목적으로 한다.

Glove에서는 조화급수(harmonic series)에  $\frac{L-\sqrt{d}+1}{L}$  따라 부여한다. 단 Glove에서는 서브 샘플링 비율(sub sampling rate) 파라미터를 적용하지는 않는다.

또 Levy and Goldberg (2014)에 따르면 Skipgram은 행렬 분해의 관점에서 해석될 수 있다. SGNS에서의 학습이 끝난 단어-문맥 임베딩 행렬의 곱  $W \cdot C^T$ 가 더 큰 행렬  $M$ 을 만든다면, SGNS의 목표  $W, C$ 의 최적화는 결국  $M$ 을 최적화하기 위함이다. 다시 말해 SGNS는  $M$ 을 분해하기 위해  $W, C$ 를 찾는 알고리즘의 일종이다. Levy et al. (2015)에 따르면 이 같은 행렬  $M$ 은 단어-문맥에 대한 shifted PMI이다.

	Glove	Skipgram with Negative Sampling
Local cost function	$l_G(w_i, c_j) = f(X_{i,j}) \times (W_i \cdot C_j^T + b_{w_i} + b_{c_j} - \log X_{i,j})$ $f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^\alpha, & x < x_{max} \\ 1, & otherwise \end{cases}$ $X_{i,j} = \#(w_i, c_j)$ $\alpha = \frac{3}{4}, \quad x_{max} = 100$	$l_s(w_i, c_j) = X_{i,j} \times \log \sigma(W_i \cdot C_j^T) + k \times \#(w_i) \cdot \frac{\#(c_j)}{\sum_w \#w} \log \sigma(-(W_i \cdot C_j^T))$
Global cost function	$\sum_{i=1}^{ V_W } \sum_{j=1}^{ V_C } l_G(w_i, c_j)$	$\sum_{i=1}^{ V_W } \sum_{j=1}^{ V_C } l_s(w_i, c_j)$
$M = W_i \cdot C_j^T$	$W_i \cdot C_j^T = X_{i,j} - b_{w_i} - b_{c_j}$	$W_i \cdot C_j^T = \text{PMI}(w_i, c_j) - \log k = X_{i,j} - \log \#(w_i) - \log \#(c_j) + \log \sum_w \#(w) - \log k$
Window	$L = \text{Maximum window size}, d = \text{disance from the focus word}$	

Weighting Scheme	$\frac{L - d + 1}{L}$	$\frac{L - \sqrt{d} + 1}{L}$
------------------	-----------------------	------------------------------

**Table 4 Glove와 SGNS의 비교**

한편, 두 모델 모두  $D$ 라는 동연한 정보를 기준으로 학습하기에 각각의 기법을 통해 도출한 임베딩 행렬은 서로 비슷한 성질을 가진다. Shi and Liu (2014)에 따르면 Glove의 목적함수가 최적화 하는 행렬은 SGNS가 최적화 하는 것과 매우 유사하다. 이는 Glove의 목적함수에서 등장하는 bias term  $b_{w_i}, b_{c_j}$ 가 SGNS의  $\log \#(w_i), \log \#(c_j)$ 와 강한 연관이 있기 때문이다. 결국 Glove와 SGNS의 가장 큰 차이는 전혀 등장하지 않는 단어에 대해 복원 비용  $l(w_i, c_j)$ 을 어떻게 처리하는가에 있다. Glove에서는 비용을 전혀 고려하지 않는 반면 SGNS에서는 무작위로 뽑은 negative sample들의 비용을 고려한다.

## 2.2. 실험소개

본 연구는 앞서 소개한 Cbow, Skipgram, Fasttext, Glove 모델로 임베딩을 만들고 LSTM과의 결합을 통해 임베딩의 성능을 측정한다. 2.2.1에서는 실험을 통해 확인하고자 하는 연구 과제를 구체적으로 명시한다. 2.2.2에서는 실험에 사용한 말뭉치 데이터를 소개하고 2.2.3에서는 각 입력 단위와 임베딩 모델의 파라미터 설정에 관한 내용을 다룬다.

### 2.2.1. 연구과제

실험을 통해 다음의 4가지를 구체적으로 밝히고자 한다:

- (1) 한국어 임베딩 학습에 적합한 입력단위는 무엇인가?

- (2) 모델간의 성능 차이와 구체적인 이유는 무엇인가?
- (3) 파라미터와 성능간에 어떤 관계가 있는가?
- (4) 파라미터를 어떻게 최적화해야 하는가?

## 2.2.2. 실험 말뭉치

본 연구에서 감정분석과 임베딩 생성을 위해 NSMC(Naver Sentiment Movie Corpus)<sup>9</sup>을 사용하였다. 네이버 무비 리뷰 코퍼스는 네이버 네티즌 평점 칼럼에서 크롤링해 만든 코퍼스로 20만개의 리뷰로 이루어져있다. 리뷰의 형식은 **Table5**와 같다. 각 리뷰당 단어수는 140을 넘지 않으며, 긍정과 부정에 따라 각각 1과 0의 레이블이 붙어 있다. 20만개 중 10만개는 긍정이고 나머지 10만개는 부정이다. 무작위 추정을 한다면 50%의 정확도를 기대할 수 있다.

Id	Document	Label
8112052	어릴때보고 지금다시봐도 재밌어요ㅋㅋ	1
9125596	가면갈수록 내용이 개판이다. 많은걸 집어넣으려다 망함	0

**Table 5 Naver Sentiment Move Corpus v1.0**

<sup>9</sup> <https://github.com/e9t/nsmc>

### 2.2.3. 임베딩 모델의 입력단위와 파라미터

실험에서 사용할 임베딩 모델은 앞서 서론에서 밝힌 바와 같이 Word2vec(Skipgram, Cbow), Glove 그리고 Fasttext이다. 모델의 입력으로 단어가 아닌 형태소를 사용할 때 어느 정도의 성능향상이 있는지 평가하고자 한다. 형태소 분석은 트위터 형태소 분석기를 (2016) 사용하여 진행하였다. 덧붙여 Fasttext모델의 경우 자모 단위와 음절 단위의 gram에 대해서 측정하고자 한다. Table6은 임베딩 모델별 입력단위를 나타낸 것이다.

각 임베딩 모델이 구체적으로 단어 벡터를 도출하는 방식은 상이하지만, 기본적으로 코퍼스를 단어와 그 단어가 등장하는 문맥의 관점에서 데이터를 가공한다는 점에서는 동일하다.<sup>10</sup> 코퍼스에 각 문장의 단어에 고려할 단어 문맥 창(chief)의 최대 크기를 조절함으로써 임베딩 성능 향상에 도움을 줄 수 있다. 본 연구에서는 문맥 창(chief)의 최대 크기로 2, 5, 7, 10을 사용한다.

최상혁 et al. (2016)에서는 50~300을 적절한 한국어 임베딩 벡터 차원의 크기(dimension size)로 보았고, 300 보다 더 큰 차원에서는 오히려 유사도 과제에 대한 성능 향상에 도움이 되지 않는다고 말한다. 본 연구에서는 임베딩 벡터의 크기로 50, 100, 300, 500, 1000을 사용한다.

최소출현 빈도(minimum occurrence) 파라미터는 특정 출현 빈도 이하의 단어를 임베딩 학습에서 배제시킴으로써 나머지 단어의 학습을 용이하게 한다. 본 연구에서는 임베딩 학습 코퍼스와 감정분석 실험 코퍼스가 동일하기에 최소출현 빈도 파라미터의 영향이 내재적 평가 방식에 비해 클 것으로 기대한다. 최소 출현 빈도로 10, 20, 50, 100을 사용한

---

<sup>10</sup> 학습의 대상이 되는 단어의 좌우에 나타나는 단어를 단어의 문맥이라 한다. 가령 “철수는 예쁜 영화의 눈을 많이 좋아한다”라는 문장에서 중앙 단어(focus word) “영화의”의 문맥은(context) 해당 단어의 좌우에 나타나는 단어들로 [“철수는”, “예쁜”, “가방을”, “많이”, “좋아한다”]이다. 이때 문맥의 범위를 특정 창(chief)의 크기를 두어 제한할 수 있다. 만약 문맥 창(chief)의 크기가 2라면 단어 “영화의”의 문맥에는 “좋아한다”가 포함되지 않는다. Pennington et al(2014)에서는 문맥 창(chief)을 기반으로 임베딩을 도출하는 방식을 국소 문맥 임베딩 기법(local context window method)로 분류한다.

다.

모델 입력단위	CBOW	Skipgram	Glove	Fasttext	
				음절	자음,모음(grapheme)
단어	[“그는”, “학생이다”]			[[“그”, “는”, “학”, “생”, “이”, “다”]	[[“ㄱ, ㅡ, ㄴ”, “ㄴ, ㅡ, ㄴ”], [“ㅎ”, “ㅡ”, “ㄱ” ...]
형태소	[“그”, “는”, “학생”, “이다”]			[ [“그”], [“는”], [“학”, “생”], [“이”, “다”]]]	[[“ㄱ, ㅡ, ㄴ”, “ㄴ, ㅡ, ㄴ”], [“ㅎ”, “ㅡ”, “ㄱ” ...]

**Table 6** “그는 학생이다”에 대한 입력 단위

## 2.3. 감정 분석

감정 분석(Sentiment Analysis)은 텍스트의 내용과 감정 레이블을 연결하는 작업이다. 가령, 영화 리뷰를 보고 단순 긍정, 부정인지 판단하는 경우가 이에 해당한다. 본 연구에서는 네이버 무비 리뷰 코퍼스(NSMC)의 리뷰들을 분류하는 과제로써 임베딩의 성능을 측정한다. 2.3.1은 감정 분석을 위한 네트워크 정의를 소개하고 2.3.2는 감정 분석 네트워크의 훈련을 위한 데이터 처리 과정을 기술한다.

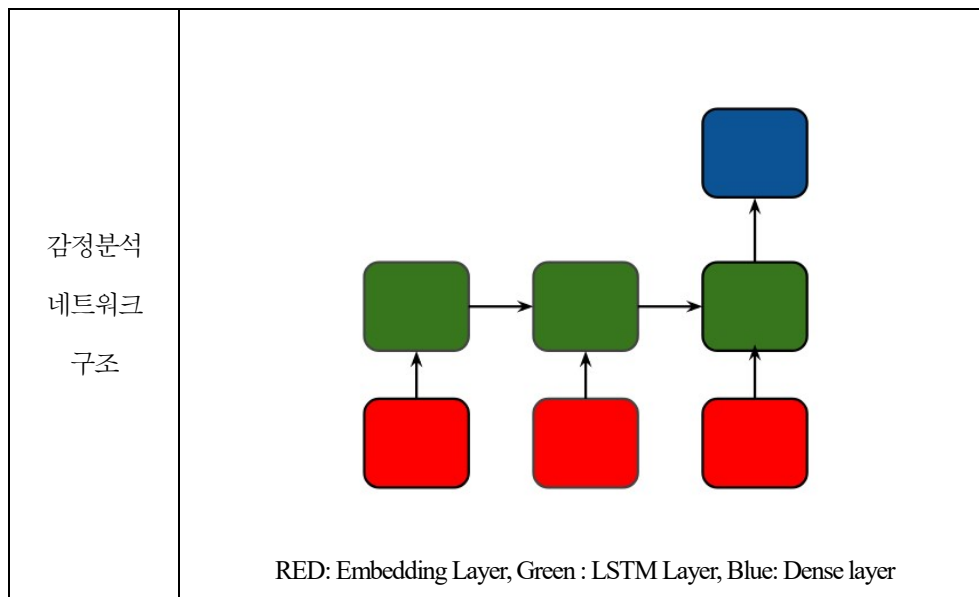
### 2.3.1. 감정 분석 네트워크 정의

본 연구에서 임베딩의 질을 평가하기 위한 감정분석 네트워크는 임베딩의 종류와 관계없이 **Table7**의 구조로 통일하였다. 감정분석을 위한 네트워크는 3개의 층으로 이루어져 있으며, 첫번째 층은 임베딩 레이어이다. 이 레이어를 학습을 마친 임베딩 행렬  $W$ 로 초기화 한다. 두번째 층은 임베딩 벡터를 입력으로 받는 LSTM(Long Short Term Memory)

레이어이다. 이 레이어에서 감정을 결정하는 뉘앙스 및 통사구조를 학습한다. 마지막 레이어는 두개의 뉴런으로 이루어진 시그모이드( $\sigma$ ) Dense 레이어이다. 이 레이어에서 최종적으로 텍스트에 대한 긍정 또는 부정을 결정한다.

중간 레이어로 LSTM을 사용하는 이유는, LSTM이 긴 시계열 데이터에 대해서 잘 학습할 수 있기 때문이다. 데이터가 길어질 경우, 바닐라 RNN은 그래디언트 소멸, 폭발 문제로부터 자유롭지 못하다. 하지만 LSTM의 경우 은닉층에 메모리 유닛이 존재해 데이터의 장기 의존성을 잘 학습할 수 있다. LSTM에는 forget, update, output 3개의 gate가 존재하는데 forget gate를 이용해 메모리에서 제거할 상태를 결정하고 update gate로부터 어떤 정보를 업데이트할 지 결정하며 최종적으로 output gate에서 이전 셀의 상태를 다음 셀로 넘겨준다.

Table 7에서  $\Gamma_f^{<t>}$ ,  $\Gamma_u^{<t>}$ ,  $\Gamma_o^{<t>}$ 는 각각 forget, update, output gate를 의미한다. 시점  $t$ 에서의 메모리 상태  $c^{<t>}$ 는  $\Gamma_f^{<t>}$ 를 거친 이전 상태의 메모리와  $\Gamma_u^{<t>}$ 를 거친 후보값  $\tilde{c}^{<t>}$ 의 합으로 정의된다.  $W_*$ 는 게이트와 관련된 가중치 행렬을,  $b_*$ 는 bias를 나타낸다.



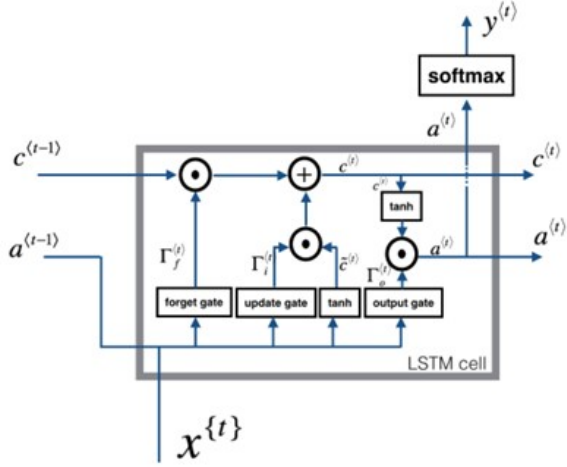
LSTM 레이어	도식 11	
	수식	$\Gamma_f^{<t>} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$ $\Gamma_u^{<t>} = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$ $\Gamma_o^{<t>} = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$ $\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$ $c^{<t>} = \Gamma_f^{<t>} \circ c^{<t-1>} + \Gamma_u^{<t>} \circ \tilde{c}^{<t>}$ $a^{<t>} = \Gamma_o^{<t>} \circ \tanh(c^{<t>})$

Table 7 감정분석 네트워크

### 2.3.2. 감정 분석 네트워크 훈련

감정분석 네트워크의 훈련을 위해 코퍼스를 (입력, 정답)의 쌍으로 재구성하였다. 제일 먼저 원시 코퍼스를 특수문자와 기호를 제외하고 입력 단위(u)로 쪼개어  $K_u =$

<sup>11</sup> LSTM By François Deloche [CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>)], from Wikimedia Commons



$[k_1, k_2, \dots, k_N]$  로 나타내고,  $K_u$ 로부터 사전  $Dict_u$ 를 만들어  $K_u$ 에 각 원소를 1-hot encoding으로 나타내었다. 또 각 문장마다 길이가 다른데, 이를 보정하기 위해 최대 길이로 zero-padding을 하였다. 긍정 또는 부정에 대한 레이블도 1-hot encoding 방식으로 표현하였다. Table 8는 데이터 처리 과정을 예문으로서 나타낸 것이다..

(1)	$K_{word}^{raw}$	[아, '더빙', '진짜', '짜증', '나네', '요', '목소리']
(2)	$K_{word}^{integer}$	[182, 132, 22, 17, 98, 16, 52]
(3)	$K_{word}^{1-hot}$	[[0,0,0,0,...1,0,0,0],[0,0,0,0,...,1,0,0] ... ]
(4)	zero_padding( $K_{word}^{1-hot}$ )	[[0,0,0,0,...1,0,0,0],[0,0,0,0,...,1,0,0] ... [0,0,...,0,]]
(5)	labels – 1hot encoding	[[0, 1]], [1, 0] ... ]

**Table 8** 훈련 데이터 처리과정

총 20만개의 샘플 중 3/4만을 훈련 샘플로 쓰고 나머지 5만개는 검증에 사용하였다. 훈련은 batch 단위로 진행하였고, batch의 크기는 128로 하였다. 과적합 방지를 위해 early stopping을 도입해, 매 epoch마다 검증 샘플에 대한 정확도를 측정해 5회안에 개선이 없으면 학습을 중단하였다. 손실 함수는 Binary Crossentropy를 옵티마이저로는 Adam(Adaptive Moment Estimation)을 사용하였다.

## 2.4. 실험 결과 해석

2.4에서는 실험 결과를 소개하고 이에 대한 심층적인 해석을 한다. 본격적인 해석에 앞서, 2.4.1에서는 실험에 사용한 임베딩을 입력 단위에 따라 나누고 중점적 분석 대상을 명시적으로 지정한다. 2.4.2에서는 입력 단위에 따른 성능을 알아보고 2.4.3에서는 모델간 성능비교를 비교한다. 마지막으로 2.4.4에서는 파라미터와 성능간의 관계를 살펴본다.

### 2.4.1. 실험 범주와 해석 방법

본 연구에서 실험한 임베딩의 유형은 입력 단위에 따라 **Table 9**에 정리하였다. 모든 파라미터 조합에 대해 실험을 한 것은 O로 표기를 하였고, 연구의 목적과 큰 연관이 없어 일부 파라미터 조합에 대해서만 실험한 것은 X로 표기하였으며 파라미터 분석에는 포함시키지 않았다.

본 연구에서는 굵게 표시한 Skipgram, Glove, Fasttext(음절)를 중점적으로 분석하였다. 더불어 O+로 표기한 Skipgram과 Fasttext의 경우 파라미터의 조합에 따른 (단어, 문맥) 튜플 전체 집합  $D$ 의 변화에 집중해 기술하였다. 이는, 본 연구에서 사용한 모델들이 모두 문맥 창 설정을 통해 도출한  $D$ 를 바탕으로 학습하기 때문이다.

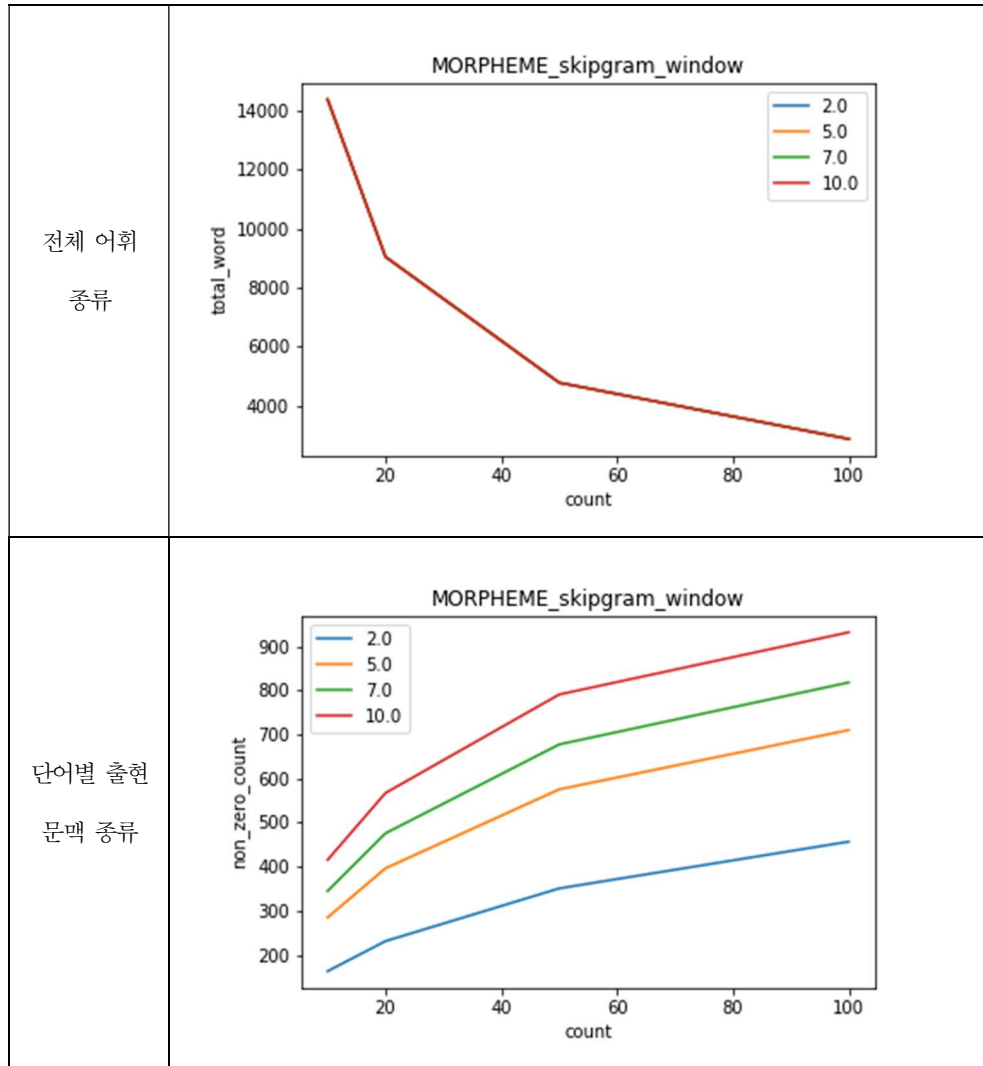
입력단위 \ 모델	CBOW	Skipgram	Glove	Fasttext	
				음절	자음, 모음(grapheme)
단어	X	<b>O+</b>	<b>O</b>	<b>O+</b>	X
형태소	O	<b>O+</b>	<b>O</b>	<b>O+</b>	O

**Table 9** 입력 단위에 따른 임베딩 종류

$D$ 의<sup>12</sup> 분포 변화는 공기(co-occurrence) 행렬  $X_{i,j}$ 을 만들어 확인한다. 예를 들어  $X^{word}$ 의 열의 개수와 셀의 값이 0이 아닌 열의 개수는 각각 전체 단어 종류와 단어별 출현 문맥 종류가 된다. 또  $\frac{1}{n} \sum_{i=1}^n \#(X_i)$ 으로 단어별 평균 출현 빈도를 구할 수 있다. **Table10**은 최소 출현 빈도에 따른 전체 어휘의 종류, 단어별 평균 출현 문맥 종류와 빈도를 나타낸 것이다. 전체 어휘 종류는 최소 출현빈도와 반비례하고 창 크기와는 무관하다. 단어별 문맥 종류의 경우 최소 출현 빈도에 따라 선형적으로 증가했다. 이는 최소 출

<sup>12</sup>  $D$ 의 경우 입력 단위와 파라미터에 따른 추가적인 표기  $D(u, w, m)$ 를 통해 구별한다. 가령 최대 크기 창 5, 최소 출현빈도 10으로 도출한 형태소 단위  $D$ 의 경우  $D^{(morpheme, w=5, m=10)}$ 으로 표기한다.

현 파라미터가 문맥 창을 만들기 이전에 적용되어 값이 커질수록 단어당 실질적 문맥창의 크기를 증가시키기 때문으로 이해된다. 창이 커질수록 단어별 평균 빈도는 증가하고 증가 폭은 최소 출현빈도 파라미터의 크기에 비례했다.



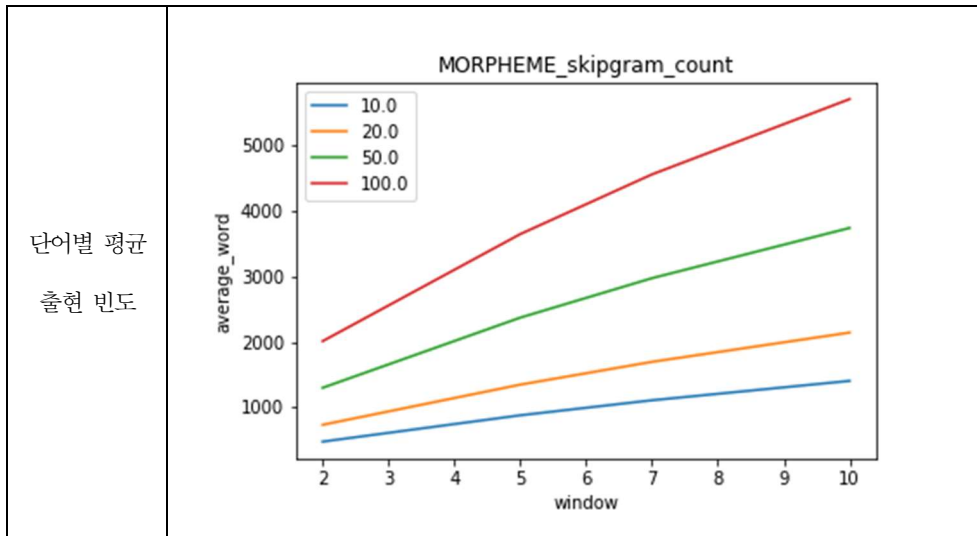
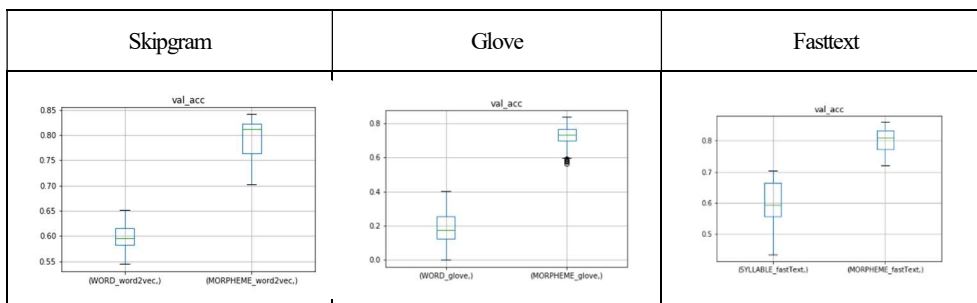


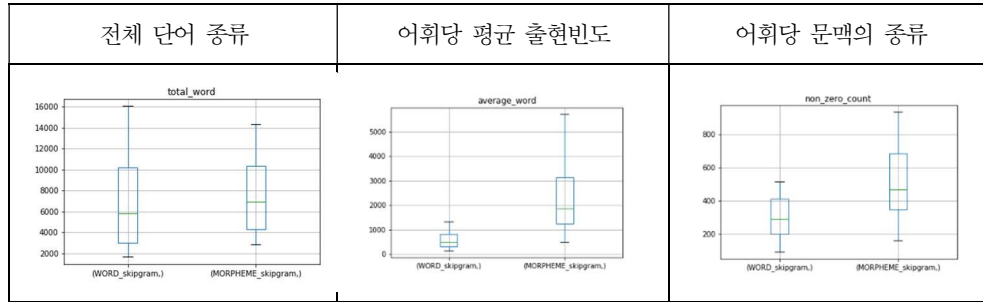
Table 10 창 크기와 최소 출현 빈도에 따른 D의 변화

## 2.4.2. 입력 단위에 따른 성능 비교

Table 11은 입력 단위에 따른 성능을 표시한 것이다. Skipgram, Glove, Fasttext 세 가지 모델 모두 형태소 단위의 입력이 성능이 더 좋았다. 입력단위가 단어에서 형태소로 바뀔 때 Table 12에서 보듯 단어별 평균 수와 문맥의 종류가 증가하기 때문으로 파악된다. 예를 들어  $D(\text{Word}, w=2, m=2)$ 에서  $D(\text{Morpheme}, w=2, m=2)$ 로 바뀔 때 전체 단어 종류는 16064에서 14382개로 줄지만(10.47%) 단어별 평균 출현 빈도는 140.48에서 479.20로 문맥 종류는 95.40에서 162.71로 증가했다.

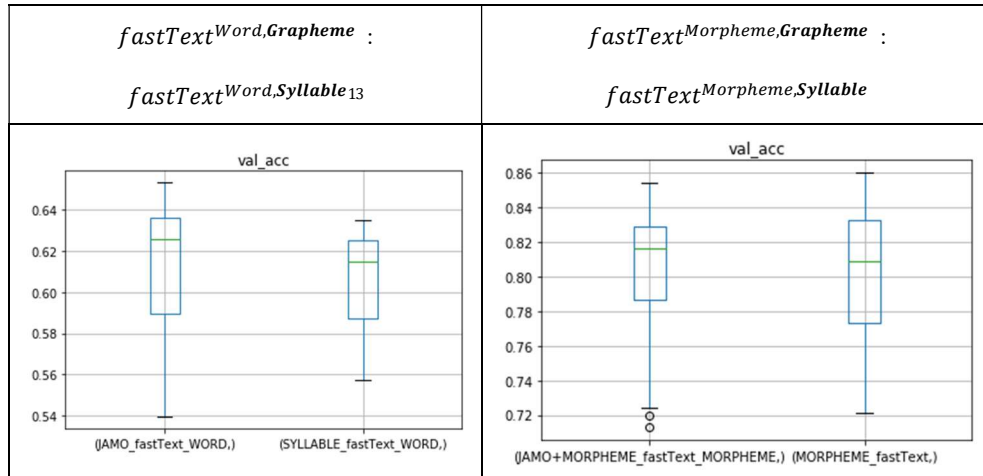


**Table 11** 형태소와 단어 단위 입력 성능 비교



**Table 12** 형태소와 단어 단위 입력 D 비교

Table13은 Fasttext의 n-gram 입력 단위로 음절과 문자소(자음, 모음)에 대해 실험한 결과이다. 실험 결과 입력 단위를 음절보다 더 잘게 쪼개는 것은, 성능 개선에 도움을 주지 않았다. 이는 문자소 단위의 n-gram이 음절 단위의 n-gram보다 평균 출현빈도가 낮고 무의미한 n-gram의 구성 비율이 음절 단위 n-gram보다 더 높기 때문으로 파악된다. 한편, 코퍼스의 크기를 늘리고, 단어 혹은 형태소를 문자소 단위의 n-gram 의 BoWs로 나타낼 때 단순합이 아닌 적응 가중치의 조합(adaptive weighted sum)으로 표현한다면 무의미한 n-gram으로 인한 노이즈를 상당부분 줄일 수 있을 것으로 기대한다.



<sup>13</sup> 임베딩을 지칭할 때,  $model^{input\ unit, (optional, n-gra\ unit)}$ 의 형식으로 나타낸다.

Table 13 n-gram 단위 비교 Grapheme vs Syllable

### 2.4.3. 모델간 성능 비교

Table 14는 모델들의 성능을 단어와 형태소의 입력단위로 구별해 나타낸 것이다. 단어와 형태소 단위의 입력 모두에서 Glove < Skipgram < Fasttext 순으로 성능이 측정되었다. 형태소 단위에서의 Glove와 Skipgram의 차이는 크지 않았지만 단어 단위의 입력에서는 Skipgram의 성능이 Glove에 비해 훨씬 좋았다. 이 둘의 차이는 전혀 등장하지 않는 단어에 대한 복원 비용  $l(w_i, c_j)$ 을 처리하는 방식에서 오는 것으로 해석된다. Glove는 복원 비용을 0으로 처리하지만 Skipgram은 무작위로 negative sample을 뽑아 관찰되지 않은 단어-문맥 쌍에 대한 비용을 고려한다. 단어 단위의 입력의 경우 상당수의 단어가 최소출현빈도 파라미터의 적용으로 인해 제거되고, 이때 기존에 0이 아닌 작은 값으로 존재했던 성분의 값이 0이된다. 즉, 단어별 평균 출현빈도가 일정 수준으로 보장되지 않는 입력은 Glove의 성능을 극감시키는 요인이 될 수 있다.

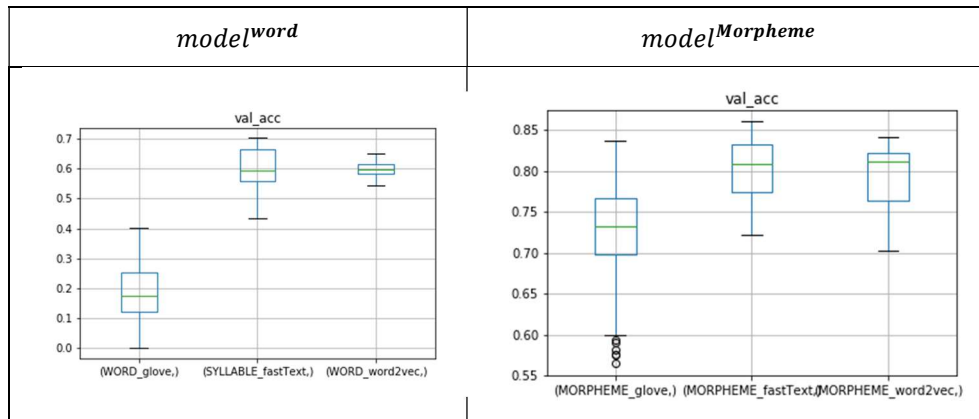


Table 14 모델간 성능 비교

Table15는 형태소 단위에서 Cbow와 Skipgram의 성능을 비교한 것이다. 실험 결과 Cbow보다 Skipgram의 성능이 더 좋았다. Cbow의 경우 목표 함수  $\frac{1}{|K|} \sum_{t=1}^{|K|} \log p_{\theta}(v_t | v_{t-\frac{s}{2}} \dots v_{t+\frac{s}{2}})$  에서 문맥 단어  $v_{t-\frac{s}{2}} \dots v_{t+\frac{s}{2}}$  부분에 대한 계산은 평균을 내는 방식  $h = \frac{1}{s} W^T (\sum_{i=1}^s x_i)$  이다. 단순 평균 방식은 문맥 단어들 간의 상대적 위치가 고려되지 않는 문제가 있다. 반면, Skipgram은 (단어, 문맥) 각 쌍에 대한 비용을 계산하기 때문에 단어간 위치 구별 문제가 발생하지 않는다.

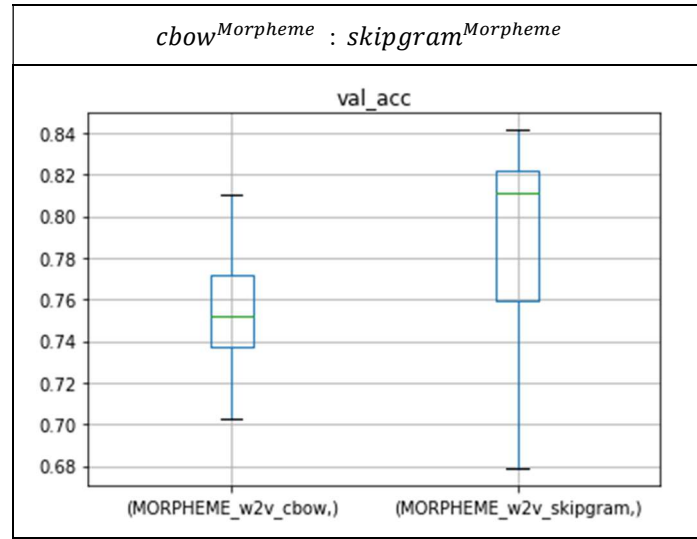


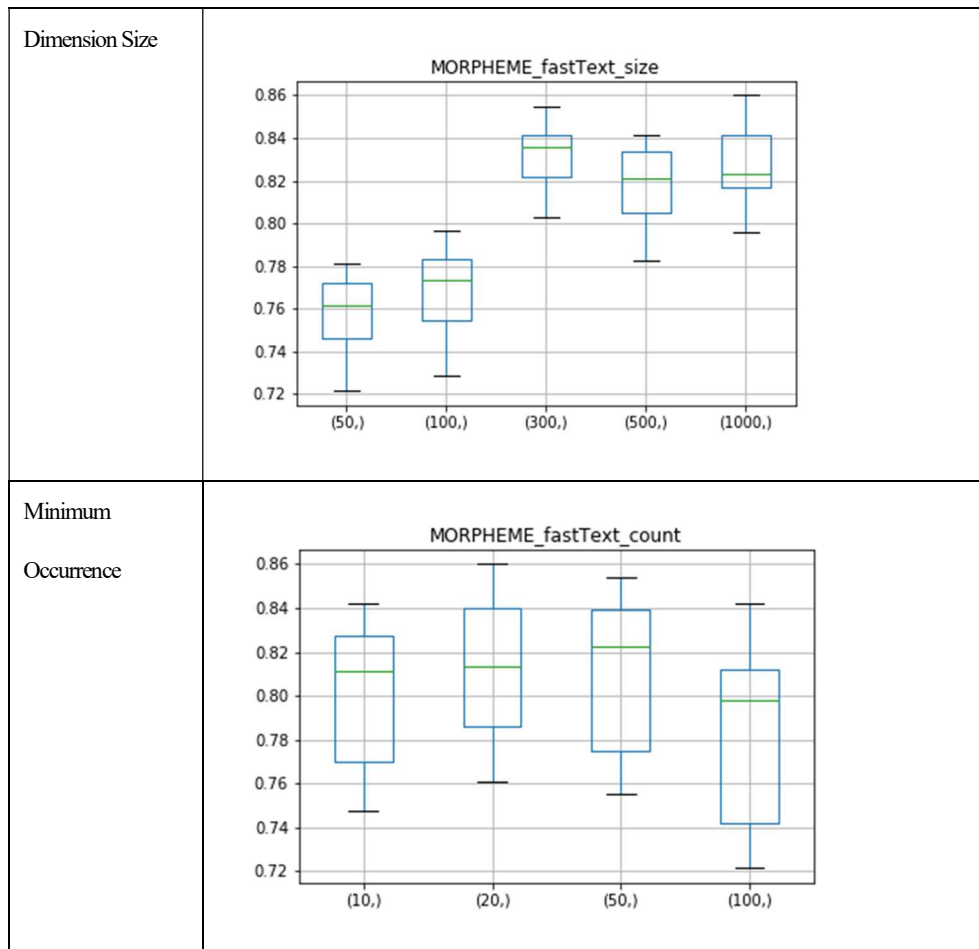
Table 15 Cbow와 Skipgram 비교

#### 2.4.4. 파라미터에 따른 성능 비교

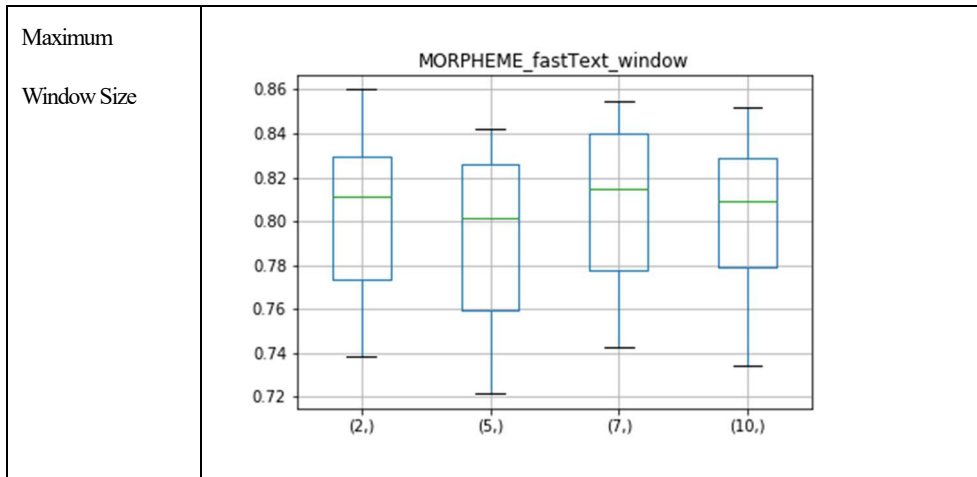
Table16은  $fastText^{morpheme, syllable}$  성능을 표시한 것이다. 도식에서 표현된 박스는 3가지 파라미터 중 한가지를 고정시키고 나머지 두 파라미터의 조합에 따른 성능을 기준으로 그린 것이다. 예를 들어 첫 번째 행의 도식은 차원의 크기를 고정시키고 창

크기와 최소 출현 빈도의 조합에 따른 성능을 표시한 것이다.

박스 플롯의 세로 폭의 크기는 고정시키지 않은 파라미터의 성능에 대한 영향력을 반영한다. 또 파라미터에 따른 박스의 중앙값(초록 선)의 변화로 개별 파라미터가 성능에 미치는 영향을 알 수 있다. 첫 번째 도식의 초록 선은 나머지 두 도식에 비해 뚜렷한 변화를 보이고 박스의 세로 폭이 좁다. 때문에 임베딩 차원의 크기 파라미터가 성능에 가장 많은 영향을 끼친다는 것을 확인할 수 있다. 나머지 두 도식의 경우 중앙값의 변화가 크지 않아, 박스 플롯만으로는 최소출현빈도와 창의크기 파라미터의 영향을 분석하기는 힘들다. 때문에 **Table17**의 도식으로 확인한다.







**Table 16** *fastText<sup>morpheme,syllable</sup>* 단일 파라미터의 영향

**Table17**의 도식은 최소출현 빈도와 최대 창의 크기의 변화에 따른 성능 변화를 3차원 공간에 나타낸 것이다. 기준 파라미터는 동일한 색으로 표현하여 나머지 파라미터에 대한 변화를 쉽게 파악할 수 있도록 하였다. 가령, 첫 번째 도식의 경우 최대창의 크기가 기준이 되어 최소출현빈도에 따른 성능 변화를 보여준다. 최소 출현빈도는 모든 차원에서  $100 < 10 < 50 < 20$  순으로 성능이 좋았다. 하지만 창 크기 파라미터는 차원의 크기마다 순위가 달라 일정하지 않았다. Fasttext의 경우 다른 창에 존재하는 단어일지라도, 같은 gram을 공유하면 학습 가능하기 때문이다. 따라서 창 크기에 따른 영향은 단어간 파라미터를 공유하지 않는 Skipgram에서 더 일관되게 나타날 것으로 예상할 수 있고 **Table18**을 통해 이를 확인하였다.

기준 size	Maximum Window	Minimum Occurrence
------------	----------------	--------------------

50	<p>50 dim/MORPHEME_fastText</p>	<p>50 dim/MORPHEME_fastText</p>
순위	[100< 10< 50< 20]	[5< 10< 2< 7]
100	<p>100 dim/MORPHEME_fastText</p>	<p>100 dim/MORPHEME_fastText</p>
순위	[100< 10< 50< 20]	[5< 2< 7< 10]
300	<p>300 dim/MORPHEME_fastText</p>	<p>300 dim/MORPHEME_fastText</p>
순위	[100< 10< 50< 20]	[5< 2< 10< 7]

500		
순위	[100<10<50<20]	[5<7<2<10]
1000		
순위	[100<10<50<20]	[10<5<7<2]

**Table 17** *fastText<sup>morpheme,syllable</sup>* 최소출현빈도-창 파라미터의 영향

**Table18**은 *Skipgram<sup>morpheme</sup>* 에 대한 최소 출현 빈도와 최대 창의 크기 파라미터에 따른 성능을 곡면 플롯으로 나타낸 것이다. 단면이 붉은 계열일수록 성능이 높게 측정된 것인데, 차원에 관계없이 붉은 단면이 형성되는 지점이 일정하다. *fastText<sup>morpheme,syllable</sup>*와는 달리 창 크기 파라미터가 성능에 미치는 영향이 일관되기 때문이다.

나아가  $D^{Morpheme}$ 의 분포와 성능이 규칙적인 연관이 있음을 예상할 수 있는데, 이는 **Table10**에서 보듯 최소출현 빈도와 창 크기 파라미터가  $D^{Morpheme}$ 의 분포에 규칙적인 변화를 일으키기 때문이다. 이는 **Table19**를 통해 확인하였다.

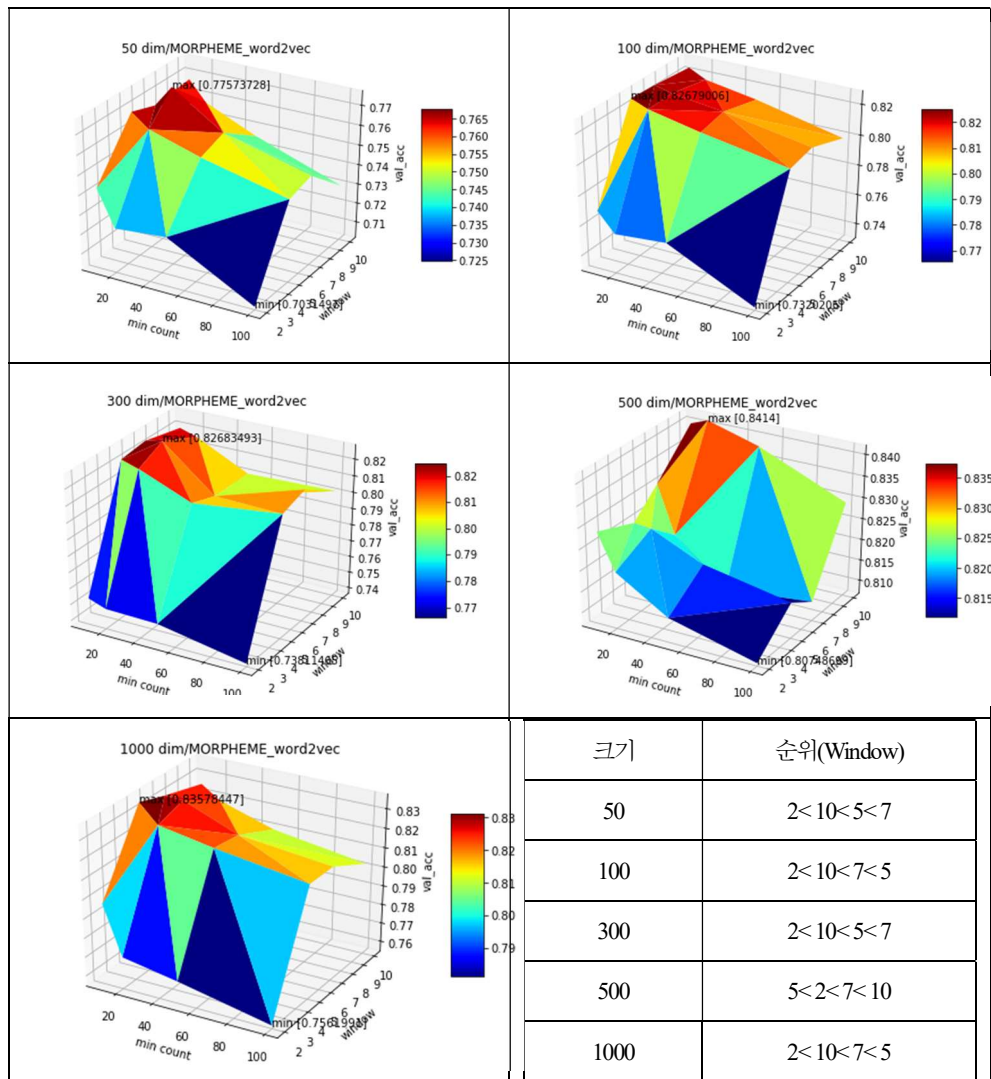


Table 18 *Skipgram<sup>morpHEME</sup>* 최소출현빈도- 창 파라미터의 영향

Table19는 최소출현빈도와 창 크기에 따른 [전체 어휘 종류/ 단어별 출현 문맥 종류/ 단어별 평균 출현 빈도]와 성능을 나타낸 것이다. 가령 차원의 크기가 300인 경우 최고점은 최소 출현빈도가 20이하에 창 크기가 5이상인 범주에 형성된다. 이때 이 범주에 속하는 (단어별 평균 출현빈도, 단어별 출현 문맥 종류) 튜플 쌍을 살펴보면 같은 색상의 조합은 비슷한 성능을 낸다. 예를 들어 표에서 2행 2열의 (1348.656, 396.151)과 1행

4열의 (1405.921, 415.299)는 같은 색상의 조합으로 성능이 비슷하다. 이 같은 사실 때문에 중복된 색상의 튜플을 제외하면 모든 파라미터 조합에 대해 살펴보지 않고도 최적점을 찾을 수 있다.



Table 19  $D^{Morpheme}$ 의 분포와 성능

Fasttext의 경우, 위에서 소개한 방식으로는 최적의 파라미터를 찾을 수 없다. 왜냐하면 일반적인 단어-문맥공기 행렬  $X_{i,j}$ 가 아니라 n-gram의 공기 행렬을 만들어야 하기 때문이다. 본 연구에서는 n-gram 공기 행렬을 통한 분석은 포함시키지는 않았다. 다만, 최소 출현빈도 파라미터가 차원의 크기에 관계없이 일정하다는 Table18에 결론을 이용하면 실

험 후보 조합은  $\#(window\_size) * \#(minimum\ count)$ 에서  $\#(window\_size)$ 로 줄어든다. 2.4의 내용을 기반으로 각 모델의 특수성을 살린 새로운 파라미터 최적화 기법은 2.5에서 구체적으로 소개한다.

## 2.5. 새로운 파라미터 최적화 방법

파라미터 최적화는 최고의 성능을 내는 파라미터의 조합을 찾는 것을 일컫는다. 본 연구에서는 격자 탐색으로 모든 파라미터에 대해 실험을 한 후 최적의 조합을 찾았는데, 이는 모델의 특수성을 고려하지 않는다는 점에서 비효율적이다. 실험에서 밝힌 내용을 활용한다면 최적화 비용을 상당히 줄일 수 있다. 2.5는 모델별 특수성을 고려해  $(S, W, M)^{14}$ 를 최적화하는 방법을 소개한다.

특수성	Fasttext	모든 차원에 공통되게 적용될 수 있는 최소출현빈도 파라미터가 존재한다.
	Skipgram	$(W_1, M_1)$ 와 $(W_2, M_2)$ 가 비슷한 $D^{Morpheme}$ 의 분포를 가진다면 비슷한 성능을 낼 것으로 기대할 수 있다.
	Glove	$D^{Word}$ 와 같은 불균형적인 분포는 좋은 성능을 기대하기 힘들다.

**Table 20** 모델의 개별 특수성

모든 모델에서  $(S, W, M)$  중  $S$ 의 범주부터 먼저 좁히는데  $S$ 가 성능에 제일 많은 영향을 미치기 때문이다. 또,  $S$ 의 경우 성능이 특정 차원을 경계로 상당한 차이를 보이며, 본 실험에서는 100이상 300이하에서 경계가 생겼다. 코퍼스의 크기와 네트워크의 푸는 문제에 따라 최적의  $S^*$ 는 달라질 수도 있기 때문에 미리 정한 특정 값이 아니라, 성능

<sup>14</sup> 편의상 임베딩 차원의 크기, 최대 창 크기, 최소출현 빈도를 각각  $S, M, W$ 로 나타낸다.

향상이 더더지는 경계  $S_{boundary}$  를 찾아 범주를  $S_{min} \sim S_{max}$  에서  $S_{boundary} \sim S_{max}$  로 좁힌다.

한편  $W, M$  의 경우에는 개별 모델의 특수성에 따라 찾는 방법이 달라진다. Fasttext는 모든 차원에 공통된 최적의 최소출현빈도 파라미터가 존재하기 때문에 임의의  $W$  를 설정해 최적의  $M^*$  을 먼저 찾은 후 나머지  $W$  후보 값들에 대해 실험한다. Skipgram과 Glove는  $W$  와  $M$  에 따른  $D^{Morpheme}$  의 분포를 이용해 후보군의 범위를 좁힌다. Table 21과 같이 단어별 평균 출현 빈도와 문맥종류를 5분위로 나눈 후 (단어별 평균 출현빈도, 단어별 문맥종류) 튜플을 만들어 중복된 튜플을 제외한다. 이 같은 방법을 쓰면 실험 후보군을 절반 가량 줄일 수 있다.

단어별 평균 출현빈도	단어당 문맥 종류	실험 후보군
		(50, 5), (100, 10)
		(20, 7), (10, 5), (50, 10), (10, 10),
[ 4      4      4      4 ]	[ 4      4      4      3 ]	(50, 7),
[ 4      4      3      3 ]	[ 4      3      3      2 ]	(100, 7)
[ 3      3      2      2 ]	[ 3      2      2      1 ]	
[ 3      1      1      0 ]	[ 2      1      1      0 ]	

Table 21 Skipgram과 Glove의 최적화 방법 예시

### 3. 결론

지금까지 다양한 입력과 파라미터 조합에 따른 임베딩을 만들고 각각의 성능을 감정 분석이라는 과제로써 평가하였다. 평가 결과를 간략히 요약하자면 (1) 한국어에서 입력 단위로는 형태소를 쓰는 것이 단어를 쓰는 것보다 월등히 좋았다. (2) 임베딩 모델 중에서는  $fastText^{morpheme, syllable}$ 가 제일 좋았으며  $grapheme$ 단위의  $n$ -gram은 성능에 방해가 되었다. (3) 차원의 크기 파라미터가 성능에 가장 큰 영향을 주었다. (4) Fasttext에서는 최적의 최소출현빈도 파라미터가 존재한다. (5) Skipgram의 경우  $D$ 의 분포와 임베딩의 성능간에는 규칙적인 연관이 있다. (6) (4), (5)의 사실을 이용하면 모든 파라미터에 대해 실험하지 않아도 최적 파라미터 조합을 찾을 수 있다.

#### 3.1. 한계점

본 연구의 한계점은 3가지 정도로 정리할 수 있다. 첫째 실험에 사용한 코퍼스의 크기가 다소 작다. 코퍼스가 좀 더 컸다면 Fasttext의  $n$ -gram 입력 단위에 대한 실험 결과가 달라졌을 수도 있다. 문자소 단위의  $n$ -gram의 경우, 음절보다 더 작은 형태소나 형태소의 경계가 음절이 아닌 경우도 학습할 수 있다는 장점이 있다. 그러나 실험에서는 음절 단위의  $n$ -gram의 최대 성능이 문자소 단위의  $n$ -gram보다 좋았다. 이는 무의미한  $n$ -gram의 비율이 음절보다 문자소 단위에서 더 높았기 때문으로 해석된다. 따라서 코퍼스의 크기를 키워 유의미한  $n$ -gram의 출현빈도를 높이고 적절한 최소 출현빈도 값을 선택함으로써 무의미한  $n$ -gram의 비율을 낮춘다면 문자소 단위의  $n$ -gram의 성능이 음절 단위보다 더 뛰어났을 수도 있다. 둘째 분류기의 종류로 LSTM 하나에 대해서만 실험하였다. MLP나 CNN 계열에 대해 성능을 측정하지 않아 모든 딥러닝 모델 유형에 대해 본 연구의 결과



를 적용하기엔 무리가 있다. 마지막으로 실험에 선정한 파라미터의 종류가 제한적이다. 본 연구에서는 세가지 모델 전부에 공통이 적용될 수 있는 S, W, M에 대해서만 알아보았다. Skipgram의 subsampling 비율과 Glove의 weighting function  $f(x)$ 의  $\alpha$  파라미터도 W와 M가 마찬가지로  $D$ 의 분포에 변화를 일으키는 요소이기에 면밀히 탐색해볼 가치가 있다.

### 3.2. 의의

본 연구의 의의는 크게 다음의 두 가지이다. 첫째, 한국어 임베딩에 대해 외재적인 평가를 한 첫 시도이다. 기존의 연구에서 한국어 임베딩은 단지 유추나 유사성 문제로만 평가되어 왔다. 그런데 임베딩은 그 자체로만 쓰이기보다는 더 큰 딥러닝 모델에 결합되어 쓰인다. 따라서 본 연구에서는 임베딩의 실질적 쓰임에 맞게 LSTM 모델에 편입시켜 감성분석 과제로써 성능을 평가하였다. 두 번째 의의는 파라미터의 변화와 성능을 심층적으로 분석했다는 것에 있다. 형태소 단위의 임베딩 모델과 파라미터 관한 기존 분석들은 단순 수치 보고 수준에서 그친다. 단순 수치가 좋게 나타났다는 방식의 기술은 실험 코퍼스의 크기나 도메인에 따라 파라미터의 최적점이 다르게 나타나기에 도움이 되지 못한다. 이에 본 연구는 파라미터의 영향을  $D^{Word/Morpheme}$ 의 변화를 중심으로 기술하였고 나아가 모델의 특수성을 고려해 파라미터 최적화 방법을 언급하였다.

## 참고문헌

- Baroni, M., & Lenci, A. (2010). Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, 36(4), 673-721.
- Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb), 1137-1155.
- Ceyda, C., & Byoung-Tak, Z. (2016). Better Word Embeddings for Korean. [한국어를 위한 개선된 워드 임베딩]. *한국정보과학회 학술발표논문집*, 2016(12), 627-629.
- Collobert, R., & Weston, J. (2008). *A unified architecture for natural language processing: Deep neural networks with multitask learning*. Paper presented at the Proceedings of the 25th international conference on Machine learning.
- Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016). *Character-Aware Neural Language Models*. Paper presented at the AAAI.
- Levy, O., & Goldberg, Y. (2014). *Neural word embedding as implicit matrix factorization*. Paper presented at the Advances in neural information processing systems.
- Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3, 211-225.
- Ling, W., Luis, T., Marujo, L., Astudillo, R. F., Amir, S., Dyer, C., . . . Trancoso, I. (2015). Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.
- Pennington, J., Socher, R., & Manning, C. (2014). *Glove: Global vectors for word representation*. Paper presented at the Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).

- Schnabel, T., Labutov, I., Mimno, D., & Joachims, T. (2015). *Evaluation methods for unsupervised word embeddings*. Paper presented at the Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.
- Shi, T., & Liu, Z. (2014). Linking GloVe with word2vec. *arXiv preprint arXiv:1411.5595*.
- Turney, P. D., & Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37, 141-188.
- Young, T., Hazarika, D., Poria, S., & Cambria, E. (2017). Recent trends in deep learning based natural language processing. *arXiv preprint arXiv:1708.02709*.
- 유홍연, & 고영중. (2017). Bidirectional LSTM CRF 기반의 개체명 인식을 위한 단어 표상의 확장. [Expansion of Word Representation for Named Entity Recognition Based on Bidirectional LSTM CRFs]. *정보과학회논문지*, 44(3), 306-313.
- 조현수, & 이상구. (2017). FastText를 적용한 한국어 단어 임베딩 (pp. 705-707).
- 최상혁. (2017). *음절 기반 한국어 단어 임베딩 모델 및 학습 기법* (국내석사학위논문), 서울대학교 대학원, 서울. Retrieved from <http://www.riss.kr/link?id=T14507496>
- 최상혁, 설진석, & 이상구. (2016). 한국어에 적합한 단어 임베딩 모델 및 파라미터 튜닝에 관한 연구. *제 28회 한글 및 한국어 정보처리 학술대회 논문집*.
- 트위터. (2016). 트위터에서 만든 오픈소스 한국어 처리기, Github, [twitter/twitter-korean-text](https://github.com/twitter/twitter-korean-text)  
Retrieved from <https://github.com/twitter/twitter-korean-text>