한국어 챗봇에서의 오류에 강건한 한국어 문장 분류를 위한 어절 단위 임베딩

최동현[a,b,O], 박일남[a], 신명철[a], 김응균[a], 신동렬[b]

카카오[a], 성균관대학교[b]

{heuristic.085, pin.park, index.shin, jason.eg}@kakaocorp.com, drshin@skku.edu

Eojeol-based Embedding for Korean Erroneous Sentence Classification in Korean Chatbot

DongHyun Choi [a,b,O], IlNam Park[a], Myeongcheol Shin[a], EungGyun Kim[a], Dong Ryeol Shin[b]

Kakao Corp[a], Sungkyunkwan University[b]

요 약

본 논문에서는 한국어 챗봇에서의 문장 분류 시스템에 대하여 서술한다. 텍스트를 입력으로 받는 한국어 챗봇의 경우, 때때로 입력 문장에 오타나 띄어쓰기 오류 등이 포함될 수 있고, 이러한 오류는 잘못된 형태소 분석 결과로 이어지게 된다. 잘못된 형태소 분석 결과로 인한 문장 분류의 오류를 줄이기 위하여, 본 논문에서는 새로운 통합 어절 임베딩 방식을 제안한다. 통합 어절 임베딩 방식의 단점을 보완하고 성능을 향상시키기 위하여, 두 가지의 말뭉치 노이즈 추가 방법이 별도로 제안되었다. 실험 결과에 따르면, 본 논문에서 제안된 시스템은 오류를 포함한 한국어 문장 분류 문제에서 기존 시스템과 비교하여 문장 단위 정확률 기준으로 23 %p의 성능 향상을 보였다.

주제어: 문장 분류, 어절 임베딩, 오타

1. 서론

챗봇에서의 자연어 이해(Natural Language Understanding) 모듈은 크게 (1) 입력 발화의 도메인(Domain)을 결정하는 도메인 분류기, (2) 결정된 도메인 내에서 사용자가 실제 실행하고자 하는 의도(Intent)를 파악하는 의도 분류기, 그리고 (3) 요구 기능을 실행하기 위한 파라미터를 얻어내는 엔티티(entity) 추출기로 이루어진다. 예를 들어, 사용자 입력 발화로 "아이유 노래 틀어줘" 가 주어질 경우, 도메인은 "음악", 음악 도메인 내에서의 사용자 의도는 "음악 재생", 그리고 실제 음악 재생을 위한 파라미터로서 가수명인 "아이유" 엔티티가 추출되어야 한다. 이 중, 도메인 분류기와 의도 분류기는 모두 주어진 문장을 여러 개의 클래스 중 하나로 분류하는 문장 분류 문제로 정의될 수 있다.

한국어는 종합어이기 때문에, 한국어 어절 하나는 두 개 이상의 형태소를 포함할 수 있다. 이로 인해, 기존 관련 연구들에서는 입력된 한국어 문장을 먼저 한국어 형태소 분석기를 이용하여 형태소 단위로 나눈 후, 이를 시스템의 입력으로 사용한다[1][2][3]. 이러한 형태소 기반 문장 분류 방식은 문법적으로 옳은 입력들에 대해서는 잘 동작하지만, 본 논문의 4.3절에 제시된 실험 결과에서 보여지듯이 오류를 포함한 문장의 경우에는 잘 동작하지 않는데, 이는 문장의 오류로 인해 잘못된 형태소 분석 결과가 얻어지기 때문이다. **표 1**은 오류가 포함된 입력 문장이 잘못된 형태소 분석 결과로 인해 오분류되는 예시이다. 문장 분류를 위해 중요한 단서인 형태소 "팟캐스트"가, 오타로 인한 형태소 분석의 오류로 인하여 무의미한 두 개의 형태소인 "파케" 와 "스트로"로 분리되었다.

표 1 잘못된 형태소 분석으로 인한 주요 단어의 손실

오류 포함된 문장	딴 파케스트로 플레이
형태소 분석 결과	딴 / 파케 / 스트로 / 플레이
교정된 문장	딴 팟케스트로 플레이
형태소 분석 결과	딴 / **팟케스트** / 로 / 플레이

사용자가 텍스트 입력 인터페이스를 이용하여 챗봇에 접근할 경우, 사용자 입력에 여러 가지 오류가 포함될 수 있다. 예를 들어, 사용자는 비슷하게 발음되는 두 단어를 혼동하거나, 키보드 상에서 가까이 위치하는 키를 잘못 타이핑하거나, 단순히 띄어쓰기를 생략할 수도 있다. 한국어 챗봇에서 위와 같이 오류를 포함한 입력이 주어졌을 때 해당 문장의 도메인 및 의도를 분석하기 위하여, 본 논문에서는 오류에 강건한 한국어 문장 분류를 위한 자모-형태소 통합 어절 임베딩을 제안한다.

통합 어절 임베딩의 기본 아이디어는, 문장 분류를 위한 시스템에 형태소 기반 임베딩 대신 어절 기반 임베딩을 입력으로 제공함으로써, 입력의 오류로 인한 잘못된 형태소 분석의 영향을 최소화하는 것이다. 이를 위해, 주어진 어절 w에 대하여 먼저 자모 기반 어절 임베딩 벡터 $\mathbf{e}_j(w)$와 형태소 기반 어절 임베딩 벡터 $\mathbf{e}_m(w)$을 각각 계산한 후, 이 두 벡터를 통합하여 통합 어절 임베딩 벡터 $\mathbf{e}_u(w)$를 도출한다. 얻어진 통합 어절 임베딩 벡터는 이후 문장 분류를 위한 합성곱 신경망(Convolutional

Neural Network)의 입력으로 사용된다. 통합 어절 임베딩 벡터를 구하는 방법은 3.1절에서 좀 더 자세히 서술한다.

어절 임베딩 벡터 방식의 오류를 포함한 한국어 문장 분류 성능을 추가적으로 향상시키기 위하여, 자모 드롭아웃과 띄어쓰기 없는 문장 생성의 두 가지 데이터 노이즈(noise) 추가 방법이 새로이 제안되었다. 4장에서 제시된 실험 결과에 따르면, 통합 어절 임베딩 벡터와 제안된 두 노이즈 추가 방법을 같이 사용하였을 경우, 기존 시스템과 비교하여 오류가 포함된 문장 분류 성능이 약 23%p 만큼 향상되었다.

2장에서는 관련 연구에 대하여 간략히 서술한다. 3장에서는 본 논문에서 제안된 알고리즘에 대하여 서술하고, 4장에서는 관련된 실험 결과를 제시한다. 5장에서는 향후 연구 및 결론에 대하여 서술한다.

2. 관련 연구

[4] 에서는 처음으로 합성곱 신경망을 문장 분류 문제에 적용하였다. [5]에서의 실험은 합성곱 신경망 기반 문장 분류 알고리즘이 기존의 SVM 기반 문장 분류 알고리즘보다 일반적으로 2~5%p 가량 더 좋은 성능을 보임을 보여주었다. [6]은 LSTM과 합성곱 신경망을 결합하여 문장 분류 문제를 해결하고자 했으나, 의미있는 성능 향상을 보이지 못했다. [7]은 여러 버전의 기 훈련된 임베딩들을 다중 채널 입력으로 고려하여 [4]를 다중 채널 입력을 사용하도록 확장하는 방식으로 문제를 해결하고자 하였다.

몇몇 관련 연구들은 한국어 문장 분류 문제를 풀고자 시도하였다. [1] 에서는 기존 GloVe[8]을 확장한 한국어 GloVe를 제안하고, 이를 [4]의 입력으로 사용하여 한국어 문장 분류 문제를 해결하고자 하였다. [9]는 띄어쓰기 오류가 있는 한국어 문장의 분류 문제를 해결하고자 하였으나, 오류의 범위가 띄어쓰기에 한정된 문제가 있다.

본 논문은 띄어쓰기 오류에 국한된 것이 아닌, 일반적인 오류를 포함한 문장에 대한 문장 분류 문제를 해결하고자 하였고, 이를 위해 기존의 형태소 임베딩 기반 방식 대신 어절 임베딩 기반 방식이 새로이 제안되었다.

3. 알고리즘

본 장에서는 제안된 알고리즘에 대하여 서술한다.

3.1 자모-형태소 통합 어절 임베딩

그림 1은 통합 어절 임베딩 벡터를 얻기 위해 제안된 네트워크 구조를 나타낸다. 주어진 어절 w에 대하여, 먼저 어절을 구성하는 자모 리스트 $J(w) = \{j_1, \dots, j_{n(w)}\}$와 형태소 리스트 $M(w) = \{(m_1, p_1), \dots, (m_{m(w)}, p_{m(w)})\}$를 도출한다. 위 수식에서 $n(w)$는 w의 자모의 개수, $m(w)$는 형태소의 개수, j_i는 i번째 자모, (m_i, p_i)는 i번째 형태소-품사 쌍을 나타낸다. **표 2**는 예시 문장의 각 어절에 대한 자모와 형태소 리스트의 예시를 보여준다.

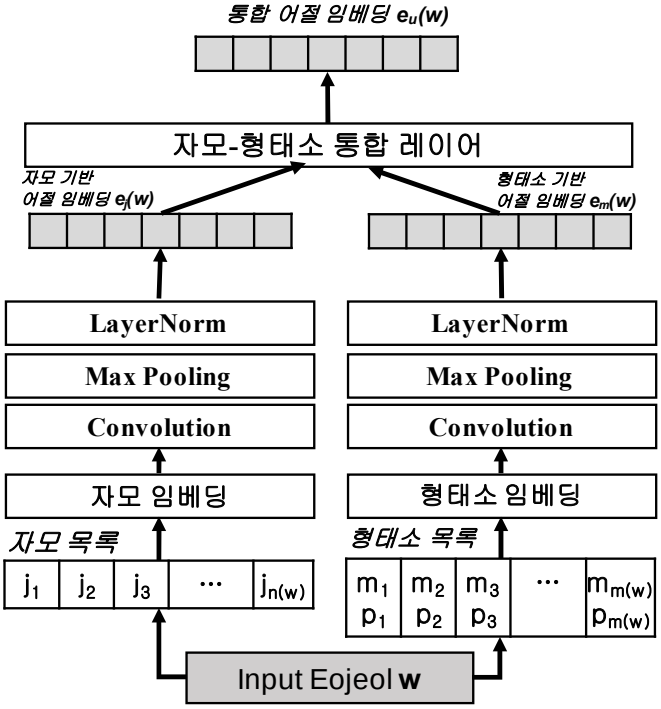

그림 1. 통합 어절 임베딩을 구하기 위한 네트워크 구조

$J(w)$와 $M(w)$는 네트워크 계산을 위하여 각각 임베딩 매트릭스 $E_J(w) = [e(j_1), \dots, e(j_{n(w)})]$ 과 $E_M(w) = [e'(m_1), \dots, e'(m_{m(w)})]$ 로 변환된다. 앞의 수식에서, $e'(m_i)$는 형태소 m_i의 임베딩 벡터와 품사 p_i의 임베딩 벡터의 연결(concatenation)으로 정의된다. 이후, 각 임베딩 매트릭스에 커널 크기 각 2, 3, 4, 5이고 필터 크기 F 인 1차원 깊이별 분리 합성곱 신경망(Depthwise separable convolution)[10]을 적용한 후, 결과 벡터에 최대값 풀링 및 layernorm[11]을 적용하여 자모 기반 어절 임베딩 벡터 $e_j(w) \in \mathbb{R}^{4F}$ 및 형태소 기반 어절 임베딩 벡터 $e_m(w) \in \mathbb{R}^{4F}$를 도출한다.

표 2 자모 / 형태소 리스트 예시

어절	자모 리스트	형태소 리스트
딴	{ㄸ,ㅏ,ㄴ}	{(딴, MM)}
팟케스트로	{ㅍ,ㅏ,ㅅ,ㅋ,ㅔ,ㅅ,ㅡ,ㅌ,ㅡ,ㄹ,ㅗ}	{(팟케스트, NNP), (로, JKB)}
플레이	{ㅍ,ㅡ,ㄹ,ㄹ,ㅔ,ㅇ,ㅣ}	{(플레이,NNG)}

통합 어절 임베딩 벡터 $e_u(w)$는 $e_j(w)$와 $e_m(w)$의 결합으로 정의된다. 본 논문에서는 두 가지 결합 방식을 제시한다. 연결(Concatenation) 방식은 $e_u(w)$ 를 $e_j(w)$와 $e_m(w)$의 연결로써 정의한다:

$$e_u(w) = [\, e_j(w); e_m(w)] \qquad (1)$$

가중치 방식은 $e_u(w)$ 를 $e_j(w)$와 $e_m(w)$의 가중치 합으로써 정의한다:

$$W = M_3 \tanh(M_1 e_j(w) + M_2 e_m(w)) \\ e_u(w) = W[e_j(w), e_m(w)] \quad (2)$$

위 수식 (2)에서, $M_1, M_2 \in \mathbb{R}^{4F \times 4F}, M_3 \in \mathbb{R}^{2 \times 4F}$는 훈련 가능한 파라미터들이고, $W \in \mathbb{R}^2, e_u(w) \in \mathbb{R}^{4F}$이다.

3.2 어절 임베딩 기반 문장 분류 네트워크 구조

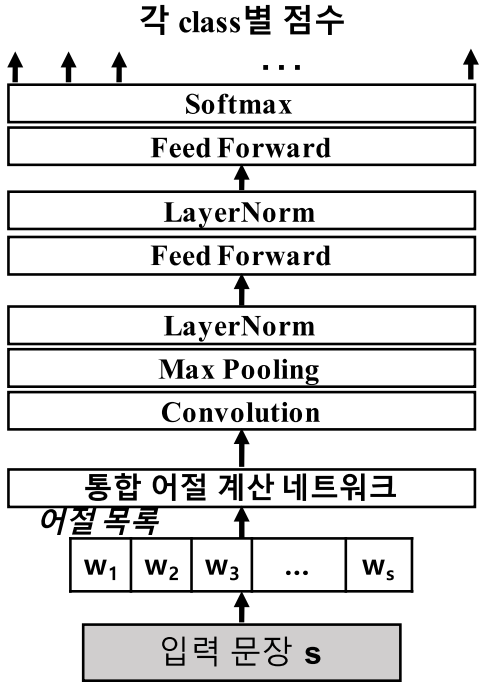

그림 2 통합 어절 임베딩 기반 문장 분류기

그림 2는 제안된 통합 어절 임베딩을 이용해 문장 분류를 수행하는 네트워크 구조를 나타낸다. 입력 한국어 문장 s는 어절 리스트 $\{w_1, \dots, w_s\}$로 생각될 수 있다. 각 어절 w_i에 대하여, 3.1절에서 제안된 방법에 따라 통합 어절 임베딩 $e_u(w_i)$가 계산된다.

통합 어절 임베딩 매트릭스 $E(s) = [e_u(w_1), \dots, e_u(w_s)]$에 커널 크기 각 1,2,3,4,5인 깊이별 분리 합성곱 신경망이 적용된다. 신경망의 결과값이 최대값 풀링 및 layernorm 레이어를 거친 후, 2개의 피드포워드(Feed forward) 레이어를 적용하고 소프트맥스(softmax) 함수를 적용함으로써 각 분류 class별 점수를 얻어낸다.

3.3 데이터 노이즈 추가

오류를 포함한 문장에 대한 분류기 성능을 좀 더 향상시키기 위하여, 두 개의 데이터 노이즈 추가 방법이 새로이 제안되었다. 첫번째로 제안된 방법인 **자모 드롭아웃**은 훈련 시에 미리 주어진 확률 *jdp*(jamo dropout probability)로 특정 자모의 임베딩 벡터값을 전부 0으로 변환하는 방법이다. 벡터의 일부 원소 값을 0으로 바꾸는 기존의 드롭아웃[12]과는 달리, 자모 드롭아웃은 랜덤하게 결정된 자모의 전체 임베딩 벡터 및, 해당 자모가 포함된 형태소의 임베딩 벡터 값을 전부 0으로 바꾼다. **자모 드롭아웃**의 목적은 첫째, 훈련 과정에서 노이즈를 추가함으로써 최종적으로 훈련된 모델이 오류를 포함한 문장을 더 잘 분류할 수 있도록 하는 것이고, 두번째로 자모를 포함하는 형태소의 임베딩 벡터 값도 같이 0으로 변환함으로써, 입력으로 들어온 형태소가 오타로 인하여 형태소 사전에 존재하지 않을 때 시스템이 자모 임베딩을 좀 더 활용하도록 학습시키기 위한 것이다.

어절은 입력 문장을 띄어쓰기 기준으로 나누어서 얻어지는 것이기 때문에, 어절 임베딩 기반 문장 분류 방식은 띄어쓰기가 없는 문장에 대하여 형태소 임베딩 기반 방식 대비 낮은 성능을 보일 것이라고 예측할 수 있다. 이러한 약점을 극복하기 위하여, **띄어쓰기 없는 문장 생성** 방법이 제안되었다. 이 방법은 훈련 말뭉치에서 *msp*(missing space probability)의 확률로 문장을 선택한 후, 해당 문장으로부터 띄어쓰기가 전부 제거된 새로운 문장을 생성하여 훈련 말뭉치에 추가하는 방식이다. **띄어쓰기 없는 문장 생성**은 훈련을 수행하기 전에 말뭉치에 1회 적용된다.

4. 실험 및 평가

본 장에서는 실험 환경 및 결과에 대하여 서술한다.

4.1 말뭉치

[1]에서 제안된 문장 분류 말뭉치는 48개 인텐트에 대하여 127,322개의 문법적으로 옳은 문장을 포함하고 있다. 포함하고 있는 인텐트의 예시로는 *날씨*(예: 오늘 날씨 어때), *운세*(예: 내일 운세 알려줘), *음악*(예: 음악 틀어줘), *팟캐스트*(예: 팟캐스트 틀어줘) 등이 있다. 각 인텐트의 문장들은 8:1:1의 비율로 훈련 데이터셋, 검증 데이터셋, 테스트 데이터셋으로 나뉘어졌다. 이 중, 테스트 데이터셋은 총 12,711개의 문장을 포함하고 있으며, 향후 서술에서는 이 테스트 데이터셋을 **WF**(Well-Formed) 말뭉치로 지칭한다.

오류를 포함한 문장에 대한 시스템 성능을 평가하기 위해, **KT**(Korean Typo) 말뭉치가 새로이 구축되었다. **KT** 말뭉치에는 총 2,070개의 문장이 포함되어 있으며, 각 문장은 최소 하나 이상의 오류를 포함하고 있다. **KT** 말뭉치는 훈련에 사용되지 않고, 모두 테스트 데이터셋으로만 이용되었다.

띄어쓰기가 없는 문장에 대한 성능을 평가하기 위해, **WF** 및 **KT** 말뭉치의 문장들을 이용하여 **SM**(Space Missing) 말뭉치가 새로이 구축되었다. **SM** 말뭉치는 **WF** 말뭉치와 **KT** 말뭉치의 문장에서 띄어쓰기가 전부 제거된 문장들로 이루어져 있으며, 총 14,781개의 문장이 포함되어 있다. **SM** 말뭉치 또한 모두 테스트 데이터셋으로만 이용되었다.

4.2 실험 환경

말뭉치에 대한 성능을 측정하기 위하여, 문장 단위 판별 정확률 *SA*(Sentence Accuracy)가 수식 (3)과 같이

정의되었다. 4.3절의 실험 결과들은 모두 *SA*값을 측정하여 제시되었다.

$$SA = \frac{\text{분류가 성공한 문장 수}}{\text{전체 문장 수}} \times 100 \qquad (3)$$

또한, 실험 결과에서 $jdp = 0.05$, $msp = 0.4$로 설정되었다. 해당 값은 $jdp = \{0.0, 0.05, 0.10, 0.15, 0.2\}$, $msp = \{0.0, 0.1, 0.2, 0.3, 0.4\}$의 값들 중 실험적으로 선택된 값들이며, 모든 설정에서 가장 좋은 성능을 보였다. 또한 각 실험에서, 동일한 설정으로 세 번의 실험이 수행되었으며, 각 실험의 테스트 성능 평균값이 본 논문에서의 최종 성능으로 제시되었다.

네트워크 훈련을 위하여 ADAM 옵티마이저[13]가 사용되었다. Learning rate의 초기 값은 0.1로 설정되었으며, 배치 크기는 128로 설정되었다. 각 epoch마다 검증 데이터셋에 대한 성능을 측정하고, 검증 데이터에 대한 *SA* 값이 감소할 때 learning rate를 절반으로 줄였다. Learning rate가 줄어들었는데도 검증 데이터에 대한 *SA* 값이 향상되지 않을 때 훈련을 종료하였다. Dropout[12]이 각 계층 사이에 적용되었으며, Dropout rate는 0.1로 설정되었다. [1]에서 제안된 기 훈련된 한국어 GloVe 임베딩 벡터가 형태소 임베딩으로 사용되었으며, 해당 값들은 훈련 과정에서 추가적으로 훈련되지 않았다. 반면, 자모 임베딩 벡터와 품사 임베딩 벡터는 네트워크 훈련 시에 같이 훈련되었다.

입력 문장의 형태소와 품사를 분석하기 위해, KHAIII[14] 오픈 소스 형태소 분석기가 이용되었다. 네트워크는 텐서플로우[15] 툴을 이용해 구현되었으며, 훈련 및 실험은 Tesla V100 GPU를 이용하여 진행되었다.

4.3 실험 결과

표 3. 통합 어절 임베딩 적용 실험 결과

시스템	WF	KT	SM
Baseline[1]	96.81	43.69	83.46
어절(C)	<u>97.05</u>	<u>46.01</u>	43.91
어절(W)	96.94	44.72	<u>44.52</u>
어절(C)+JD	97.49	<u>67.31</u>	<u>46.46</u>
어절(W)+JD	<u>97.52</u>	66.23	44.78
어절(C)+SG	<u>97.16</u>	<u>52.31</u>	<u>88.31</u>
어절(W)+SG	97.03	52.13	88.02
어절(C)+ALL	<u>97.44</u>	67.00	<u>90.76</u>
어절(W)+ALL	97.38	<u>67.60</u>	90.69

표 3은 통합 어절 임베딩 기반 문장 분류 시스템에 대한 성능 평가 결과를 보여 준다. **표 3**에서, **어절(C)**는 연결 방식 기반 통합 어절 임베딩, **어절(W)**는 가중치 방식 기반 방식 통합 어절 임베딩을 나타낸다. 또한, JD는 **자모 드롭아웃** 방식의 적용, SG는 **띄어쓰기 없는 문장 생성** 방법의 적용을 나타낸다. ALL은 JD와 SG가 같이 적용되었음을 나타낸다. 기존 형태소 임베딩 기반 문장 분류 시스템과의 결과 비교를 위하여, [1]의 시스템을 동일 데이터를 이용해 훈련 및 테스트하였다. 해당 시스템의 성능은 Baseline으로 표기되었다.

실험 결과에서 보여지듯이, 통합 어절 임베딩 기반 문장 분류 시스템을 제안된 두 개의 데이터 노이즈 추가 방법과 같이 사용할 경우 문장 분류 성능이 크게 향상되었다. Baseline 시스템과 비교하였을 때, WF 말뭉치에 대한 성능은 0.63%p, KT 말뭉치에 대한 성능은 24.3%p가 향상되었다. 3.3절에서 예측되었듯이, 통합 어절 임베딩을 사용할 경우 SM 말뭉치에 대한 성능은 크게 저하되나, 제시된 실험 결과에서 보여지듯이 **띄어쓰기 없는 문장 생성** 방법을 사용함으로써 이러한 통합 어절 임베딩의 단점을 극복할 수 있었다. **띄어쓰기 없는 문장 생성** 방법을 사용한 시스템 **어절(C)+SG**의 SM 말뭉치에 대한 성능은 88.31% 인데, 이는 Baseline 시스템의 83.46% 보다 오히려 4.85%p 더 향상된 것이다.

자모 드롭아웃 방법의 사용은 KT 말뭉치에 대한 시스템 성능을 크게 향상시킬 뿐만 아니라, WF 말뭉치에 대한 성능도 어느 정도 향상시키는 것이 실험적으로 확인되었다. 이는 3.3절에서 제안된 바와 같이, JD 방법의 사용이 자모 기반 어절 임베딩 벡터와 형태소 기반 어절 임베딩 벡터 중 어느 쪽에 더 중점을 두어야 할지를 학습하는 데 도움이 됨을 보여준다.

어절 임베딩에 의한 성능 향상과 두 데이터 노이즈 추가 방법에 의한 성능 향상을 구분하기 위하여, 통합 형태소 임베딩이 제안되었다. 통합 형태소 임베딩은 3.1절의 통합 어절 임베딩 네트워크 계산 과정에서, 어절 대신 형태소가 네트워크의 입력으로 주어지고 네트워크 출력 또한 형태소 단위로 바뀌는 것을 제외하고는 통합 어절 임베딩과 동일하게 정의된다. **표 4**는 동일한 데이터 노이즈 추가 방법이 적용되었을 때, 통합 형태소 임베딩과 통합 어절 임베딩간의 분류 성능 비교가 제시되었다.

표 4. 통합 어절 임베딩과 통합 형태소 임베딩의 비교

시스템	WF	KT	SM
형태소(C)	96.76	43.88	<u>83.33</u>
형태소(W)	95.92	<u>46.94</u>	83.07
어절(C)	<u>97.05</u>	46.01	43.91
형태소(C)+JD	97.15	56.52	<u>86.97</u>
형태소(W)+JD	95.25	58.15	85.13
어절(C)+JD	<u>97.49</u>	<u>67.31</u>	46.46
형태소(C)+SG	96.71	50.73	86.04
형태소(W)+SG	95.51	<u>53.19</u>	85.84
어절(C)+SG	<u>97.16</u>	52.31	<u>88.31</u>
형태소(C)+ALL	97.16	58.76	88.86
형태소(W)+ALL	95.62	60.08	87.62
어절(C)+ALL	<u>97.44</u>	<u>67.99</u>	<u>90.76</u>

표 4에서 보여지듯이, 데이터 노이즈 추가 방식을 사용할 경우 통합 형태소 임베딩 기반 시스템의 성능 또한 향상되나, 향상폭이 통합 어절 임베딩 기반 시스템에 비해 매우 낮다. 형태소 임베딩 기반 시스템의 경우, 두 가

지의 데이터 노이즈 추가 방식을 전부 사용 시 KT 말뭉치에 대한 성능 향상 폭은 12~13%p 정도이나, **어절(C)**의 경우 21.30%p가 향상된다. 결과적으로, 모든 데이터 노이즈 추가 방식이 적용된 경우 어절 임베딩 기반 문장 분류 시스템은 형태소 임베딩 기반 문장 분류시스템에 비하여 KT 말뭉치에서 7~9%p 정도의 성능 향상을 보인다.

마지막으로, 통합 어절 임베딩에서 자모 기반 어절 임베딩과 형태소 기반 어절 임베딩의 효과를 구분하기 위한 실험이 수행되었다. **표 5**의 실험 결과에서 **자모-어절**은 통합 임베딩 벡터 $e_u(w) = e_j(w)$, **형태소-어절**은 $e_u(w) = e_m(w)$ 으로 정의된 시스템이다.

표 5 자모 기반 어절 임베딩과 형태소 기반 어절 임베딩 효과 비교

시스템	WF	KT	SM
자모-어절	95.96	64.40	35.54
형태소-어절	96.81	43.54	45.82
어절(C)	97.05	46.01	43.91
자모-어절+JD	95.97	68.45	33.45
형태소-어절+JD	97.04	49.71	44.26
어절(C)+JD	97.49	67.31	46.46
자모-어절+SG	95.91	65.17	88.88
형태소-어절+SG	96.78	50.11	84.77
어절(C)+SG	97.16	52.31	88.31
자모-어절+ALL	95.97	69.29	89.21
형태소-어절+ALL	96.88	50.65	71.83
어절(C)+ALL	97.44	67.99	90.76

자모-어절 시스템은 형태소 분석 결과와 관련이 없기 때문에 형태소 분석 결과의 오류로부터는 자유로우나, 대신 형태소 단위로 기 훈련된 정보를 사용할 수 없다. 따라서, **자모-어절+ALL** 시스템은 KT 말뭉치에서 **어절(C)+ALL** 시스템보다도 1.30%p 높은 성능을 보였으나, WF 말뭉치에서는 1.47 %p 낮아진 성능을 보여주었다. 반면에, **형태소-어절** 시스템은 형태소 분석 오류가 발생하였을 때 오류를 보충하여 사용할 수 있는 다른 정보가 없기 때문에, 오류를 포함하는 문장에 대한 성능이 비교적 낮다. **표 5**의 실험 결과는, 통합 어절 임베딩이 문법적으로 옳은 문장에 대해서는 형태소 기반 어절 임베딩의 정보에 좀 더 의존하고, 오류가 포함된 문장에 대해서는 자모 기반 어절 임베딩의 정보를 더 많이 사용함을 보여준다.

4.4 문법 교정기 사용 실험 결과

오류를 포함한 문장을 분류하기 위해 가장 간단히 생각할 수 있는 방법은 기 존재하는 문법 교정기를 사용하는 것이다. 본 논문에서는, [16]의 알고리즘을 이용하여 구현된 카카오 맞춤법 검사 및 문법 교정기[17]를 이용해 오류를 포함한 문장을 맞춤법에 맞게 수정 후, 본 논문에서 제시된 문장 분류 시스템을 이용해 분류를 시도해 보았다. **표 6**에 실험 결과가 제시되었다. **QC**(Query Corrector) 플래그는 테스트 문장이 문장 분류 시스템에 입력되기 전, 문법 교정기를 이용해 교정되었음을 의미한다.

표 6. 문법 교정기를 적용하였을 때의 성능 평가

	KT		SM	
	QC	-	QC	-
형태소(C)	47.00	43.88	83.96	83.33
형태소(C)+JD	58.45	56.52	87.26	86.97
형태소(C)+SG	53.40	50.73	86.28	86.04
형태소(C)+ALL	60.74	58.76	89.00	88.86
어절(C)	48.92	46.01	81.77	43.91
어절(C)+JD	67.92	67.31	86.51	46.46
어절(C)+SG	54.61	52.31	87.08	88.31
어절(C)+ALL	68.78	67.99	90.80	90.76

실험 결과에서 보여지듯이, 문법 교정기는 오류를 포함한 문장의 분류 성능을 거의 향상시키지 못했다. 문법 교정기 적용 후, KT 말뭉치에서의 각 문장 분류 시스템 성능은 약 1~3%p 정도 향상되는 데 그쳤다. 이는 문법 교정기 자체가 형태소 분석 결과에 크게 의존하기 때문으로, 오타로 인해 형태소 분석기 결과에 오류가 발생한 상황에서는 문법 교정기로 인한 성능 향상이 제한적이게 되기 때문이다. 반면에, 제한된 통합 어절 임베딩 기반 시스템은 형태소 분석 오류로 인한 영향을 최소화할 수 있어, 형태소 임베딩과 문법 교정기를 같이 사용한 시스템에 비해 더 좋은 성능을 보였다.

5. 결론

본 논문에서는 기존의 형태소 임베딩 대신, 새로운 통합 어절 임베딩을 사용하여 오류를 포함한 한국어 문장을 분류하는 시스템에 대하여 서술되었다. 어절 임베딩의 단점을 극복하고 훈련 데이터에 자동으로 노이즈를 추가하기 위하여, 두 데이터 노이즈 추가 방법이 추가적으로 제안되었다. 실험 결과에 따르면, 제안된 시스템은 기존 관련 연구에서 제안된 형태소 기반 문장 분류 시스템과 비교하여 오류를 포함한 문장 분류 문제에서 24.3%p, 문법적으로 옳은 문장을 분류하는 문제에서 0.63%p의 성능 향상을 보였다.

실험 결과는 여러 가지 다른 단위의 임베딩을 하나로 통합하고 데이터에 자동으로 노이즈를 추가함으로써, 오류를 포함한 문장 분류 성능을 크게 향상시킬 수 있음을 보여주었다. 향후 연구로, 더 많은 종류의 임베딩들을 하나로 통합시키는 방법 및 새로운 데이터 노이즈 추가 방법에 대한 연구를 진행하고 있다.

참고문헌

[1] 최동현, 박일남, 임재수, 백슬예, 이미옥, 신명철, 김응균, 신동렬, 한국어 대화 엔진에서의 문장 분류, 제 30회 한글 및 한국어 정보처리 학술발표 논문집,

pp. 210-214, 2018.
[2] 박일남, 최동현, 신명철, 김응균, GloVe와 최대 엔트로피 모델을 이용한 한국어 문장 분류 시스템, 제 30회 한글 및 한국어 정보처리 학술발표 논문집, pp. 522-526, 2018.
[3] K. Oh, D. Lee, H. Choi and J. Hur, A model of Korean sentence similarity measurement using sense-based morpheme embedding and RNN sentence encoding, IEEE International Conference on Big Data and Smart Computing, pp. 430-433, 2017.
[4] Y. Kim, Convolutional neural networks for sentence classification, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1746-1751, 2014.
[5] Y. Zhang and B. Wallace, A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification, arXiv preprint arXiv:1510.03820, 2015.
[6] C. Zhou, C. Sun, Z. Liu and F. C. M. Lau, A c-lstm neural network for text classification, arXiv preprint arXiv:1511.08630, 2015.
[7] W. Yin and H. Schutze, Multichannel variable-size convolution for sentence classification, Proceedings of the Nineteenth Conference on Computational Natural Language Learning, pp. 204-214, 2015
[8] J. Pennington, R. Socher and C. D. Manning, Glove: Global vectors for word representation, Proceedingsof Empirical Methods in Natural Language Processing, pp. 1532-1543, 2014.
[9] 박근영, 김경덕, 강인호, 띄어쓰기 오류에 강건한 문장 압축 기반 한국어 문장 분류, 제 30회 한글 및 한국어 정보처리 학술발표 논문집, pp. 600-604, 2018
[10] F. Chollet, Xception: Deep learning with depthwise separable convolutions, IEEE Conference on Computer Vision and Pattern Recognition(CVPR), pp. 1800-1807, 2017.
[11] J. L. Ba, J. R. Kiros and G. E. Hinton, Layer normalization, arXiv preprint arXiv:1607.06450, 2016.
[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research, 15(1), pp. 1929-1958, 2014.
[13] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations, Vol. 5, 2015.
[14] https://github.com/kakao/khaiii
[15] https://www.tensorflow.org
[16] M. D. Kernighan, K. W. Church and W. A. Gale, A spelling correction program based on a noisy channel model, Proceedings of the 13th conference on Computational linguistic, Vol. 2, pp. 205-210, 1990.
[17] https://alldic.daum.net/grammar_checker.do