*** 쇄별 판권 날짜**

- **1 쇄(2019 년 9 월)**
- **2 쇄(2019 년 10 월)**
- **3 쇄(2020 년 2 월)**
- **3.5 쇄(2020 년 10 월)**

*** p.56 부터 2 쇄/3 쇄 페이지 달라집니다.**

(3 쇄부터 2 페이지씩 추가)

P. 32~33 (1 쇄 p. 32/ 2 쇄 p. 32/ 3 쇄 p. 32)

구체적으로는 첫 번째 단어 벡터 + 두 번째 단어 벡터 - 세 번째 단어 벡터를 계산해 보는 것이다. 그림 1-3 처럼 아들 + 딸 - 소녀 = 소년이 성립하면 성공적인 임베딩이라고 볼 수 있다. (중략) 단어 1 + 단어 2 - 단어 3 연산을 수행한 벡터와 코사인 유사도가 가장 높은 단어들이 네 번째 열의 단어들이다.
>
구체적으로는 첫 번째 단어 벡터 - 두 번째 단어 벡터 + 세 번째 단어 벡터를 계산해 보는 것이다. 그림 1-3 처럼 아들 - 딸 + 소녀 = 소년이 성립하면 성공적인 임베딩이라고 볼 수 있다. (중략) 단어 1 - 단어 2 + 단어 3 연산을 수행한 벡터와 코사인 유사도가 가장 높은 단어들이 네 번째 열의 단어들이다.

P. 35~36 (1 쇄 p. 35~36/ 2 쇄 p. 35~36/ 3 쇄 p. 35~36)

그림 1-4 를 다음으로 대체

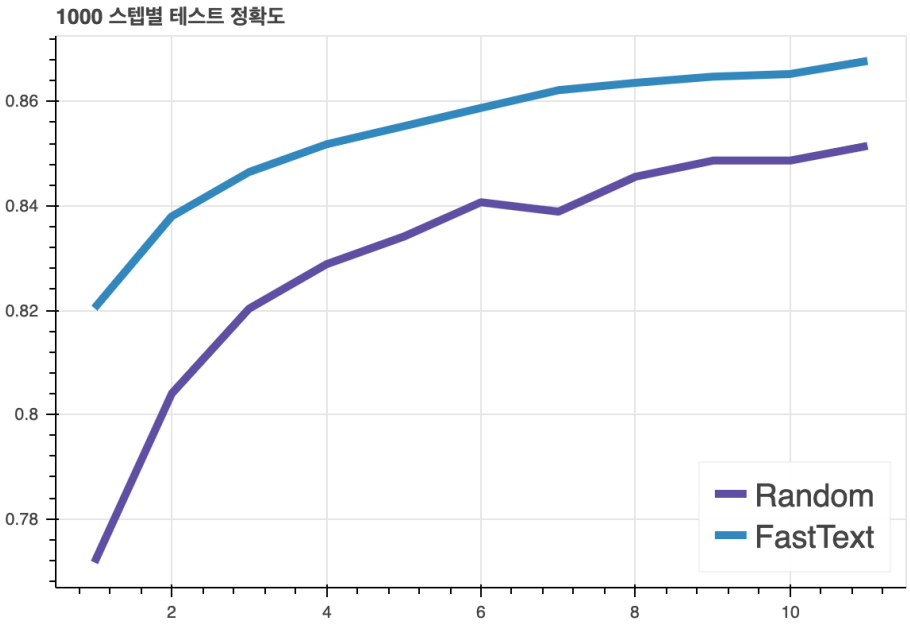
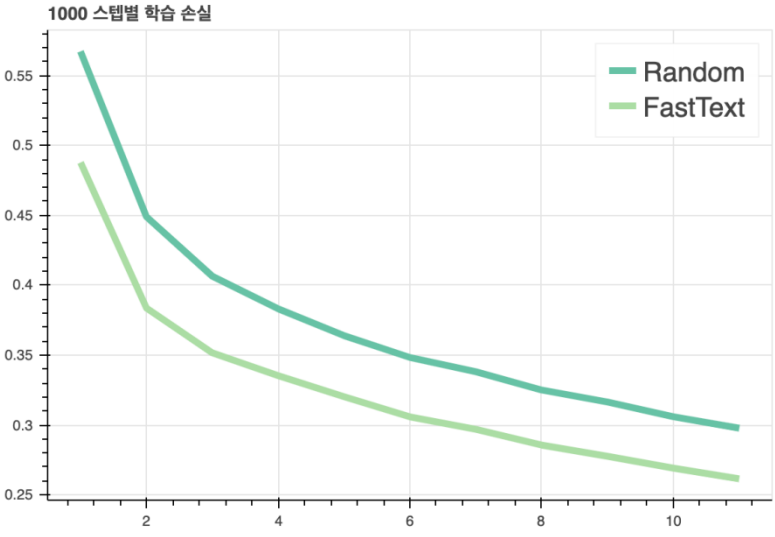


그림 1-5 를 다음으로 대체



P. 40 (1 쇄 p. 40/ 2 쇄 p. 40/ 3 쇄 p. 40)

우리가 풀고 싶은 자연어 처리의 구체적 문제들을 다운스트림 태스크(downstearm task)라고 한다.
>
우리가 풀고 싶은 자연어 처리의 구체적 문제들을 다운스트림 태스크(downstream task)라고 한다.
sequence-to-sequenece
> sequence-to-sequence

P. 48 (1 쇄 p. / 2 쇄 p. / 3 쇄 p.)

AWS 를 사용하지 않고 개인 컴퓨터에서 GPU 환경 도커 컨테이너를 띄우려면 nvidia-docker 가 반드시 설치돼 있어야 한다.
>
AWS 를 사용하지 않고 개인 컴퓨터에서 GPU 환경 도커 컨테이너를 띄우려면 nvidia-docker 가 반드시 설치돼 있어야 한다(nvidia-docker 는 윈도우/맥을 지원하지 않으므로 우분투 운영체제에서 nvidia-docker 를 설치하기를 권한다).

코드 1-2 를 다음과 같이 수정

```
docker run -it --rm --runtime=nvidia ratsgo/embedding-gpu bash
>
docker run -it --rm --gpus all ratsgo/embedding-gpu bash
```

P. 55 (1 쇄 p. 55/ 2 쇄 p. 55/ 3 쇄 p. 55)

- 띄어쓰기 : 실수인, 초월수는, 모두, 무리수이다.
어휘 집합(vocabulary)은 말뭉치에 있는 모든 문서를 문장으로 나누고 여기에 토크나이즈를 실시한 후 중복을 제거한 토큰들의 집합이다.
>
- 띄어쓰기 : 실수인, 초월수는, 모두, 무리수이다.

한편 형태소 분석(morphological analysis)이란 문장을 형태소 시퀀스로 나누는 과정을 가리킨다. 우리 책에서는 형태소와 토큰을 구분하고 있지 않으므로 별도의 설명이 없다면 형태소 분석, 토크나이즈, 토큰화 등을 같은 의미로 쓴 것이다. 이들 용어 차이에 큰 의미를 두지 않아도 된다. 다만 한국어는 토큰화와 품사 판별(Part Of Speech Tagging)이 밀접한 관계를 가지고 있고, 대부분의 오픈소스 한국어 형태소 분석기가 품사 판별까지 수행하고 있어 광의로 해석하는 쪽에서는 형태소 분석을 '토큰화 + 품사 판별'로 이해하고 있다는 점 역시 참고하자.
어휘 집합(vocabulary)은 말뭉치에 있는 모든 문서를 문장으로 나누고 여기에 토크나이즈를 실시한 후 중복을 제거한 토큰들의 집합이다.

P. 62 (1 쇄 p. 62/ 2 쇄 p. 62/ 3 쇄 p. 64)

TF-IDF 행렬 구축 튜토리얼은 4.1 절, '잠재 의미 분석'에서 설명한다.
>
TF-IDF 행렬 구축 튜토리얼은 4.4 절, '잠재 의미 분석'에서 설명한다.

표 2-3 과 관련 설명을 다음으로 교체

표 2-3 을 보면 '어머니'라는 명사는 '사랑 손님과 어머니'라는 문서에서 의존 명사 '것'보다 TF-IDF 값이 크다.
'것'만 봐서는 해당 문서의 주제를 추측하기 어렵지만, 어머니가 나왔다면 문서 주제를 예측하기가 상대적으로 수월하다.

(중략)

표 2-3 TF-IDF 행렬
구분	메밀꽃 필 무렵	운수 좋은 날	사랑 손님과 어머니	삼포 가는 길
어머니	0.066	0.0	0.595	0.0
것	0.2622	0.098	0.145	0.0848

표 2-3 처럼 TF-IDF 를 적용하면 '것' 같이 정보성이 없는 단어들은 그 가중치가 줄게 돼 불필요한 정보가 사라진다.

P. 72 (1 쇄 p. 72/ 2 쇄 p. 72/ 3 쇄 p. 74)

언어학자들이 제시하는 품사 분류 기준은 기능(function), 의미(meaning), 형태(form) 등 세 가지다.
>
학교문법에 따르면 품사 분류 기준은 기능(function), 의미(meaning), 형식(form) 등 세 가지다.

P. 83 (1 쇄 p. 83/ 2 쇄 p. 83/ 3 쇄 p. 83)

코드 3-2 를 다음으로 교체

```
from gensim.corpora import WikiCorpus, Dictionary
from gensim.utils import to_unicode

in_f = "/notebooks/embedding/data/raw/kowiki-latest-pages-articles.xml.bz2"
out_f = "/notebooks/embedding/data/processed/processed_wiki_ko.txt"
output = open(out_f, 'w')
wiki = WikiCorpus(in_f, tokenizer_func=tokenize, dictionary=Dictionary())
i = 0
for text in wiki.get_texts():
      output.write(bytes(' '.join(text), 'utf-8').decode('utf-8') + '₩n')
      i = i + 1
      if (i % 10000 == 0):
            print('Processed ' + str(i) + ' articles')
output.close()
print('Processing complete!')
```

P. 98 (1 쇄 p. 98/ 2 쇄 p. 98/ 3 쇄 p. 100)

표 3-3
Kkma : 아버지/NNG, 가방/NNG, 에/JKM, 들어가/VV, 시/EPH, ㄴ다/EFN'
Mecab : 아버지/NNG, 가/JKS, 방/NNG, 에/JKB, 들어가/VV, 신다/EP+EC

P. 101 (1 쇄 p. 101/ 2 쇄 p. 101/ 3 쇄 p. 103)

/notebooks/embedding/preprocessmecab-user-dic.csv 에 붙여 썼으면 하는 단어를 그림 3-15 와 같이 추가해주면 된다.

>

/notebooks/embedding/preprocess/mecab-user-dic.csv 에 붙여 썼으면 하는 단어를 그림 3-15 와 같이 추가해주면 된다.

그림 3-15

가우스전자,,,,NNP,*,T,가우스전자,*,*,*,*,*

서울대입구역,,,,NNP,*,T,서울대입구역,*,*,*,*,*

>

가우스전자,,,,NNP,*,F,가우스전자,*,*,*,*,*

서울대입구역,,,,NNP,*,T,서울대입구역,*,*,*,*,*

P. 106 (1 쇄 p. 106/ 2 쇄 p. 106/ 3 쇄 p. 108)

사용 방법은 코드 3-27 과 같으며 그 결과는 그림 3-17 과 같다.

>

사용 방법은 코드 3-27 과 같으며 그 결과는 그림 3-17 과 같다. FullTokenizer 분석 결과 '##'로 시작하는 토큰은 이전 토큰과 같은 어절에 있다는 뜻이다.

```
from bert.tokenization import FullTokenizer
>
from models.bert.tokenization import FullTokenizer
```

P. 117 (1 쇄 p. 117/ 2 쇄 p. 117/ 3 쇄 p. 119)

네 번째 단어가 running 인 데이터를 뽑아보면 그림 4-5 와 같다.

>

문장에서 네 번째 단어가 running 인 3-gram 을 뽑아보면 그림 4-5 와 같다.

P. 118 (1 쇄 p. 118/ 2 쇄 p. / 3 쇄 p. 120)

다섯 번째 단어가 in 인 데이터를 뽑아보면 그림 4-6 이 된다.

>

문장에서 다섯 번째 단어가 in 인 3-gram 을 뽑아보면 그림 4-6 이 된다.

P. 122 (1 쇄 p. 122/ 2 쇄 p. 122/ 3 쇄 p. 124)

수식 4-6 에서 f(w_i)란 해당 단어가 말뭉치에서 차지하는 비율(해당 단어 빈도/어휘 집합 크기)을 의미한다.
>
수식 4-6 에서 U(w_i)란 해당 단어의 유니 그램 확률(해당 단어 빈도/전체 단어 수)을 의미한다.
수식 4-6 : f(w_i)를 U(w_i)로 대체

P. 123 (1 쇄 p. 123/ 2 쇄 p. 123/ 3 쇄 p. 125)

서브샘플링 확률은 수식 4-8 과 같다. (중략) 만일 f(w_i)가 0.01 로 나타나는 빈도 높은 단어(예컨대 조사 은/는)는 위 식으로 계산한 P(w_i)가 0.9684 나 돼서 해당 단어가 가질 수 있는 100 번의 학습 기회 가운데 3~4 번 정도는 학습에서 제외하게 된다. 반대로 등장 비율이 적어 P(w_i)가 0 에 가깝다면 해당 단어가 나올 때마다 빼놓지 않고 학습을 시키는 구조다.
>
서브샘플링 확률은 수식 4-8 과 같다. f(w_i)는 w_i 의 빈도를 가리키며 t 는 하이퍼파라메터이다. Mikolov et al. (2013b)은 t 를 10^{-5}로 설정했다. (중략) 만일 f(w_i)가 0.01 로 나타나는 빈도 높은 단어(예컨대 조사 은/는)는 위 식으로 계산한 P_{subsampling}(w_i)가 0.9684 나 돼서 해당 단어가 가질 수 있는 100 번의 학습 기회 가운데 96 번 정도는 학습에서 제외하게 된다. 반대로 등장 비율이 적어 P_{subsampling}(w_i)가 0 에 가깝다면 해당 단어가 나올 때마다 빼놓지 않고 학습을 시키는 구조다.

P. 125 (1 쇄 p. 125/ 2 쇄 p. 125/ 3 쇄 p. 127)

모델 학습이 완료되면 U 만 d 차원의 단어 임베딩으로 쓸 수도 있고, U+V 행렬을 임베딩으로 쓸 수도 있다. 혹은 U, V 를 이어 붙여 2d 차원의 단어 임베딩으로 사용할 수도 있다.
>
모델 학습이 완료되면 U 만 d 차원의 단어 임베딩으로 쓸 수도 있고, U+V^T 행렬을 임베딩으로 쓸 수도 있다. 혹은 U, V^T 를 이어 붙여 2d 차원의 단어 임베딩으로 사용할 수도 있다.

P. 126 (1 쇄 p. 126/ 2 쇄 p. 126/ 3 쇄 p. 128)

코드 4-4 의 첫번째 라인을 다음으로 교체

```
cd /notebooks/embedding
>
cd /notebooks/embedding
mkdir -p data/word-embeddings/word2vec
```

P. 127 (1 쇄 p. 127/ 2 쇄 p. 127/ 3 쇄 p. 129)

```
파이썬 콘솔에서 코드 4-5 를 실행하면 기준 단어와 코사인 유사도가 가장 높은 10 개와 그 유사도가 출력된다.
>
파이썬 콘솔에서 코드 4-5 를 실행하면 기준 단어와 코사인 유사도가 가장 높은 5 개와 그 유사도가 출력된다.
```

P. 143 (1 쇄 p. 143/ 2 쇄 p. 143/ 3 쇄 p. 145)

코드 4-21 앞에 다음 두 개 라인 추가

```
cd /notebooks/embedding
mkdir -p data/word-embeddings/lsa
```

P. 144 (1 쇄 p. 144/ 2 쇄 p. 144/ 3 쇄 p. 146)

```
수식 4-20 을 보면 단어 i, j 각각에 해당하는 벡터 U_i, V_j 사이의 내적 값과 두 단어 동시 등장 빈도(A_ij)의 로그 값 사이의 차이가 최소화될수록 학습 손실이 작아진다.
>
수식 4-20 을 보면 단어 i, j 각각에 해당하는 벡터 U_i, V_j 사이의 내적 값과 '두 단어 동시 등장 빈도의 로그 값(log_{A_ij})' 사이의 차이가 최소화될수록 학습 손실이 작아진다.
```

P. 145 (1 쇄 p. 145/ 2 쇄 p. 145/ 3 쇄 p. 147)

이밖에 U+V, U 와 V 를 이어 붙여 임베딩으로 사용하는 것도 가능하다.
>
이밖에 U+V^T, U 와 V^T 를 이어 붙여 임베딩으로 사용하는 것도 가능하다.

P. 146 (1 쇄 p. 146/ 2 쇄 p. 146/ 3 쇄 p. 148)

코드 4-26 의 마지막 라인을 다음으로 교체

models/glove/build/glove -save-file data/word-embeddings/glove/glove.vecs -threads 4 (하략)
>
models/glove/build/glove -save-file data/word-embeddings/glove/glove -threads 4 (하략)

P. 147 (1 쇄 p. 147/ 2 쇄 p. 147/ 3 쇄 p. 149)

f(xij)가 클수록 Ui,Uj 벡터 간 내적값이 실제 PMI 값과 좀 더 비슷해야 학습 손실이 줄어든다.
>
f(xij)가 클수록 Ui,Vj 벡터 간 내적값이 실제 PMI 값과 좀 더 비슷해야 학습 손실이 줄어든다.

P. 148 (1 쇄 p. 148/ 2 쇄 p. 148/ 3 쇄 p. 150)

두 단어가 한 번도 등장하지 않았을 때 PMI 는 음의 무한대(infinity)로 발산하기 때문에...
>
두 단어가 한 번도 동시에 등장하지 않았을 때 PMI 는 음의 무한대(infinity)로 발산하기 때문에...

P. 155 (1 쇄 p. 155/ 2 쇄 p. 155/ 3 쇄 p. 157)

표 4-7 과 그 설명을 다음으로 대체

>
단어 유추 평가(word analogy test)는 1 장 서론에서 이미 소개했던 것처럼 갑과 을의 관계는 정과 병의 관계와 같다 는 의미론적 유추에서 단어 벡터 간 계산을 통해 갑 - 을 + 병 이라는 질의에 정 을 도출해낼 수 있는지를 평가한다. 이 평가에서는 갑 - 을 + 병 에 해당하는 벡터에 대해 코사인 유사도가 가장 높은 벡터에 해당하는 단어가 실제 정 인지를 확인한다. 이동준 외(2018)는 구글에서 만든 Google analogy 를 참고해 단어 유추 평가를 위한 데이터셋을 구축해 공개했다. 평가셋 총 건수는 420 개다. 이 절에서는 이 데이터로 평가를 진행한다. 표 4-7 과 같다.

4.7.3 단어 유추 평가

단어 유추 평가word analogy test는 1장 서론에서 이미 소개했던 것처럼 갑과 을의 관계는 정과 병의 관계와 같다는 의미론적 유추에서 단어 벡터 간 계산을 통해 갑 - 을 + 병이라는 질의에 정을 도출해낼 수 있는지를 평가한다. 이 평가에서는 갑 - 을 + 병에 해당하는 벡터에 대해 코사인 유사도가 가장 높은 벡터에 해당하는 단어가 실제 정인지를 확인한다. 이동준 외(2018)는 구글에서 만든 Google analogy를 참고해 단어 유추 평가를 위한 데이터셋을 구축해 공개했다. 평가셋 총 건수는 420개다. 이 절에서는 이 데이터로 평가를 진행한다. 표 4-7과 같다.

표 4-7 단어 유추 평가 데이터셋(이동준 외, 2018)

갑	을	병	정
대한민국	서울	일본	도쿄
대한민국	서울	중국	베이징
대한민국	서울	미국	워싱턴
대한민국	서울	영국	런던
대한민국	서울	프랑스	파리

P. 159 (1 쇄 p. 159/ 2 쇄 p. 159/ 3 쇄 p. 161)

코드 4-41 단어 벡터 간 유사도 시각화
>
코드 4-41 단어 벡터 간 유사도 시각화 (python)

P. 161 (1 쇄 p. 161/ 2 쇄 p. 161/ 3 쇄 p. 163)

예컨대 주제가 정치인 상황에서 사퇴, 경제에서 인수 같은 단어가...
>
'경제' 단어도 정치, 사퇴, 인수와 마찬가지로 블록 서식 처리

P. 162 (1 쇄 p. 162/ 2 쇄 p. 162/ 3 쇄 p. 164)

수식 4-27 에서 0 을 볼드체(zero vector)로 변경
그런데 Arora et al. (2016)에 따르면 임의의 상수 C 에 관해 max(C + c g) = g/||g||임이 성립한다고 한다.
>
그런데 Arora et al. (2016)에 따르면 임의의 상수 C 에 관해 argmax(C + c g) = g/||g||임이 성립한다고 한다.
수식 4-28 맨 마지막 줄에서 argmax 제거
따라서 우리는 우리가 관찰하고 있는 문장의 등장 확률을 최대한으로 높이는 주제 벡터를 수식 4-29 와 같이 정리할 수 있다.
>
따라서 우리는 우리가 관찰하고 있는 문장의 등장 확률을 최대한으로 높이는 주제 벡터를 수식 4-29 와 같이 정리할 수 있다(벡터 크기 정규화는 생략).

P. 163 (1 쇄 p. 163/ 2 쇄 p. 163/ 3 쇄 p. 165)

Continous Bag of Words Model 이라는 취지에서 이렇게 이름을 붙였다.
>
Continuous Bag of Words Model 이라는 취지에서 이렇게 이름을 붙였다.

P. 168 (1 쇄 p. 168/ 2 쇄 p. 168/ 3 쇄 p. 170)

코드 4-46 에 정의된 예측 단계에서는 테스트 문장을...
>
코드 4-47 에 정의된 예측 단계에서는 테스트 문장을...

P. 170 (1 쇄 p. 170/ 2 쇄 p. 170/ 3 쇄 p. 172)

가중 임베딩을 쓸 경우 임베딩을 만들 때 사용한 말뭉치의 통계량을 확인해야 하기 때문에 embedding_corpus_path 에 해당 말뭉치 경로를 추가로 입력해줘야 한다.
>
가중 임베딩을 쓸 경우 임베딩을 만들 때 사용한 말뭉치의 통계량을 확인해야 하기 때문에 embedding_corpus_path 에 해당 말뭉치 경로를 추가로 입력해줘야 한다. corpus_mecab.txt 를 만드는 방법은 코드 4-2 를 참고하자.

P. 181 (1 쇄 p. 181/ 2 쇄 p. 181/ 3 쇄 p. 183)

그림 5-3 왼쪽의 리스트는 해당 마크다운 문서의 영문 제목명과 그에 해당하는 문서 임베딩의 코사인 유사도를 가리킨다. (중략) 그림 5-3 의 우측 그림은 실제 블로그 에서 Related Posts 로 제시하고 있는 문서 목록이다.
>
그림 5-3 오른쪽의 리스트는 해당 마크다운 문서의 영문 제목명과 그에 해당하는 문서 임베딩의 코사인 유사도를 가리킨다. (중략) 그림 5-3 의 좌측 그림은 실제 블로그 에서 Related Posts 로 제시하고 있는 문서 목록이다.

P. 184 (1 쇄 p. 184/ 2 쇄 p. 184/ 3 쇄 p. 186)

수식 5-1 에서 ₩Sigma_{t=k}^{T-k} > ₩Sigma_{t=k}^{T-1}
y_i 를 만드는 방식은 이렇다. 이전 k 개 단어들을 W 라는 단어 행렬에서 참조한 뒤 평균을 취하거나 이어 붙인다. 여기에 U 라는 행렬을 내적하고 바이어스 벡터 b 를 더해준 뒤 소프트맥스를 취한다. 여기에서 U 의 크기는 어휘 집합 크기 x 임베딩 차원 수다.
>
y 를 만드는 방식은 이렇다. 이전 k 개 단어들을 W 라는 단어 행렬에서 참조한 뒤 평균을 취하거나 이어 붙인다. 여기에 U 라는 행렬을 내적하고 바이어스 벡터 b 를 더해주면 된다. 이렇게 만든 y 에 소프트맥스를 취하면 확률 벡터가 된다. U 의 크기는 어휘 집합 크기 x 임베딩 차원 수다.

P. 187 (1 쇄 p. 187/ 2 쇄 p. 187/ 3 쇄 p. 189)

```
from gensim.models.doc2vec import Togged Document
>
from gensim.models.doc2vec import TaggedDocument
```

P. 194 (1 쇄 p. 194/ 2 쇄 p. 194/ 3 쇄 p. 196)

LSA 의 학습이 끝나면 우리는 표 5-3 의 행 벡터들을 각 문서가 3 차원으로 표현된 문서 임베딩으로 활용할 수 있다.
>
LDA 의 학습이 끝나면 우리는 표 5-3 의 행 벡터들을 각 문서가 3 차원으로 표현된 문서 임베딩으로 활용할 수 있다.
이는 θd 와 z_(d,n)에 동시에 영향을 받는다. 의미는 이렇다. 직전 예시에서 z_(3,1) (3 번째 문서 첫 번째 단어의 주제)가 토픽 2 라고 가정하자. 이제 θ2 를 보자. 그러면 w_(3,1) (3 번째 문서 첫번째 단어)은 돈이 될 가능성이 높다. 토픽 2 의 단어분포 가운데 돈이 0.313 으로 가장 높기 때문이다.
>
이는 ϕk 와 z_(d,n)에 동시에 영향을 받는다. 의미는 이렇다. 직전 예시에서 z_(3,1) (3 번째 문서 첫 번째 단어의 주제)가 토픽 2 라고 가정하자. 이제 ϕ2 를 보자.

그러면 w_(3,1) (3 번째 문서 첫번째 단어)은 돈이 될 가능성이 높다. 토픽 2 의 단어분포 가운데 돈이 0.313 으로 가장 높기 때문이다.

P. 195 (1 쇄 p. 195/ 2 쇄 p. 195/ 3 쇄 p. 197)

d 번째 문서 i 번째 단어(z_(d,i))가 실제 j 번째 토픽이 될 확률을 깁스 샘플링을 적용해 구하면 수식 5-4 와 같다.
>
d 번째 문서 n 번째 단어(w_(d,n))가 실제 j 번째 토픽이 될 확률을 깁스 샘플링을 적용해 구하면 수식 5-4 와 같다.

P. 198 (1 쇄 p. 198/ 2 쇄 p. 198/ 3 쇄 p. 200)

표 5-8 단어-주제 행렬
>
표 5-8 단어-토픽 행렬

P. 201 (1 쇄 p. 201/ 2 쇄 p. 201/ 3 쇄 p. 203)

코드 5-21 LDA 학습 스크립트 (python)
>
코드 5-21 LDA 학습 스크립트 (bash)

P. 207 (1 쇄 p. 207/ 2 쇄 p. 207/ 3 쇄 p. 209)

ELMo 의 original 모델에서는 코드 5-27 처럼 컨볼루션 연산을 시행한다.
>
ELMo 의 original 모델에서는 코드 5-27 의 필터들을 사용하여 컨볼루션 연산을 시행한다.
코드 5-27 에서는 컨볼루션 필터를 일곱 종류를 쓰고 있다.
>
코드 5-27 에서는 컨볼루션 필터를 일곱 종류를 모두 쓰고 있다.

P. 210 (1 쇄 p. 210/ 2 쇄 p. 210/ 3 쇄 p. 212)

수식 2

C(x,W_c) > C(x,W_C)

P. 211 (1 쇄 p. 211/ 2 쇄 p. 211/ 3 쇄 p. 213)

그림 5-18 을 다음으로 교체

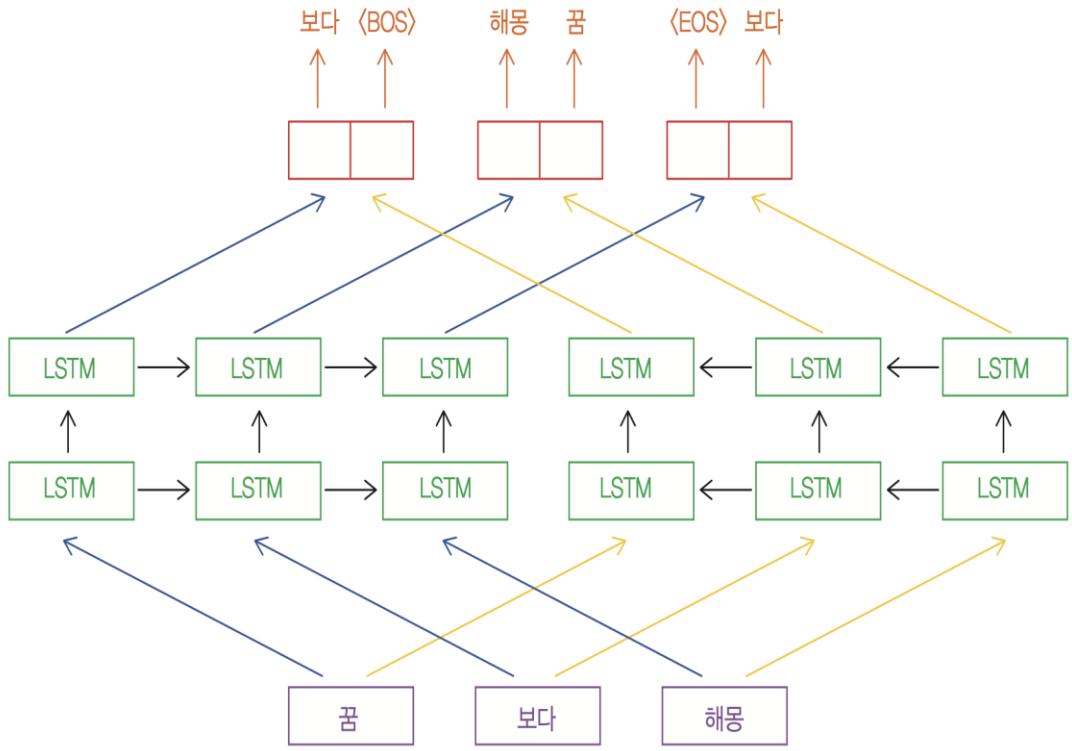


그림 5-18 ELMo 양방향 LSTM 및 출력 레이어

P. 212 (1 쇄 p. 212/ 2 쇄 p. 212/ 3 쇄 p. 214)

수식 5-5 에서 h_{k,j}^{LM}는 k 번째 토큰의 j 번째 레이어의 양방향, 역방향 LSTM 히든 벡터를 이어 붙인 벡터를 가리킨다.
>
수식 5-5 에서 h_{k,j}^{LM}는 k 번째 토큰의 j 번째 레이어의 순방향, 역방향 LSTM 히든 벡터를 이어 붙인 벡터를 가리킨다.

P. 213 (1 쇄 p. 213/ 2 쇄 p. 213/ 3 쇄 p. 215)

gamma^task 는 해당 태스크가 얼마나 중요한지 뜻하는 가중치를 의미한다.
>
gamma^task 는 ELMo 벡터의 크기를 스케일해 해당 태스크 수행을 돕는 역할을 한다.
따라서 ELMo 모델은 입력 문장의 토큰 수만큼의 임베딩들을 반환하게 된다.
>
ELMo 모델은 입력 문장의 토큰 수만큼의 토큰 임베딩들을 반환하게 된다.
한편 L 은 양방향 LSTM 레이어 수를 가리킨다. 보통 L=2 로 설정한다. j=0 일 때는 문자 단위 컨볼루션 레이어 출력, j=1 일 때는 양방향 LSTM 레이어의 첫번째 레이어 출력, j=2 일 때는 두번째 레이어 출력을 의미한다.

P. 214 (1 쇄 p. 214/ 2 쇄 p. 214/ 3 쇄 p. 216)

ELMo 모델에서 어휘 집합이 쓰이는 경우는 입력 단계와 예측 단계다.
>
ELMo 모델에서 어휘 집합이 쓰이는 경우는 프리트레인 입력 단계와 프리트레인 예측 단계다.

P. 215 (1 쇄 p. 215/ 2 쇄 p. 215/ 3 쇄 p. 217)

어휘 집합은 예측 단계에도 쓰인다.
>
어휘 집합은 프리트레인 예측 단계에도 쓰인다.

P. 216 (1 쇄 p. 216/ 2 쇄 p. 216/ 3 쇄 p. 218)

cell_clip, proj_clip 은 LSTM 셀과 셀 사이로 전파되는 그래디언트의 제한 크기를 뜻한다. 이보다 크면 해당 값으로 바꿔 그래디언트 익스플로딩(gradient exploding)을 막는다.
>
cell_clip, proj_clip 은 LSTM 셀의 값(value)의 제한 크기를 뜻한다. 이보다 크면 해당 값으로 바꿔 그래디언트 문제를 막는다.

P. 217 (3 쇄)

3 쇄의 다음 문장을 아래와 같이 교체

어휘 집합은 프리트레인 예측 단계에서도 쓰인다. 쓰인다.
>
어휘 집합은 프리트레인 예측 단계에서도 쓰인다.

P. 219 (1 쇄 p. 219/ 2 쇄 p. 219/ 3 쇄 p. 221)

그림 5-20 을 양분해 하단 멀티헤드 어텐션(Multi-Head Attention)은 5.6.1, 5.6.2 절에서, 상단 피드포워드 네트워크(feedforward network)는 5.6.3 절에서 설명한다.
>
그림 5-20 을 양분해 하단 멀티헤드 어텐션(Multi-Head Attention)은 5.5.1, 5.5.2 절에서, 상단 피드포워드 네트워크(feedforward network)는 5.5.3 절에서 설명한다.

P. 222 (1 쇄 p. 222/ 2 쇄 p. 222/ 3 쇄 p. 224)

개별 소프트맥스 값이 지나치게 작아지는 것을 방지
>
소프트맥스의 그래디언트가 지나치게 작아지는 것을 방지

P. 223 (1 쇄 p. 223/ 2 쇄 p. 223/ 3 쇄 p. 225)

5.6.1 절에서 미리 만들어놓은 쿼리(Q), 키(K), 값(V)에 Scaled Dot-Product 를 h 번 수행한다.
>
5.5.1 절에서 미리 만들어놓은 쿼리(Q), 키(K), 값(V)에 Scaled Dot-Product 를 h 번 수행한다.

P. 225~226 (1 쇄 p. 225~226/ 2 쇄 p. 225~226/ 3 쇄 p. 234)

Pointwise Feedforward Networks
>
Position-wise Feedforward Networks

P. 230 (1 쇄 p. 230/ 2 쇄 p. 230/ 3 쇄 p. 232)

전체 학습 데이터 토큰의 15%를 마스킹한다
>
학습 데이터 한 문장 토큰의 15%를 마스킹한다

P. 232 (1 쇄 p. 232/ 2 쇄 p. 232/ 3 쇄 p. 234)

BERT 가 사용하는 트랜스포머 블록에서 원조 트랜스포머와 가장 큰 차이점을 보이는 대목은 Pointwise Feedforward Networks 쪽이다.
>
BERT 가 사용하는 트랜스포머 블록에서 원조 트랜스포머와 가장 큰 차이점을 보이는 대목은 Position-wise Feedforward Networks 쪽이다.

P. 246 (1 쇄 p. 246/ 2 쇄 p. 246/ 3 쇄 p. 248)

한편 6 장에서 언급하는 코드의 대부분은 tuning_utils.py 에 정의돼 있다.
>
한편 6 장에서 언급하는 코드의 대부분은 tune_utils.py 에 정의돼 있다.

P. 249 (1 쇄&2 쇄)

코드 6-3 의 7 행에서 if is_traning: > 삭제

P. 254~255 (1 쇄 p. 254~255/ 2 쇄 p. 254~255/ 3 쇄 p. 256~257)

코드 6-7 을 다음으로 교체

```
attention_score = tf.nn.softmax(tf.contrib.layers.fully_connected(inputs=H,
num_outputs=1, activation_fn=None))
>
attention_score = tf.nn.softmax(tf.contrib.layers.fully_connected(inputs=H,
num_outputs=1, activation_fn=None), axis=1)
```

P. 258 (1 쇄 p. 258/ 2 쇄 p. 258/ 3 쇄 p. 260)

```
이보다 짧으면 [PAD]에 해당하는 ID(len(self.vocab) - 1)를 붙여 해당 길이로
맞춰준다.
>
이보다 짧으면 [PAD]에 해당하는 ID(len(self.vocab) + 1)를 붙여 해당 길이로
맞춰준다.
```

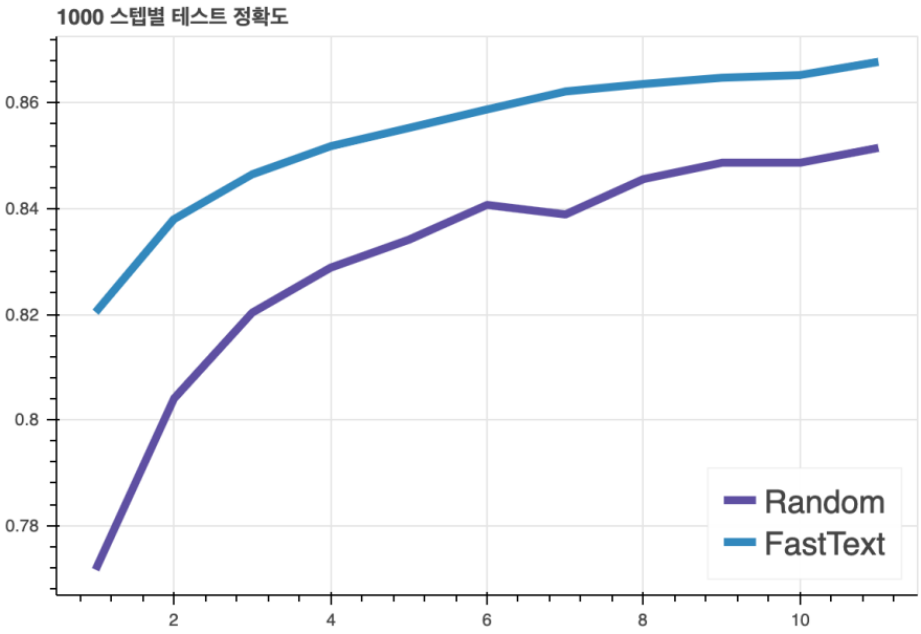
P. 259 (1 쇄 p. 259/ 2 쇄 p. 259/ 3 쇄 p. 261)

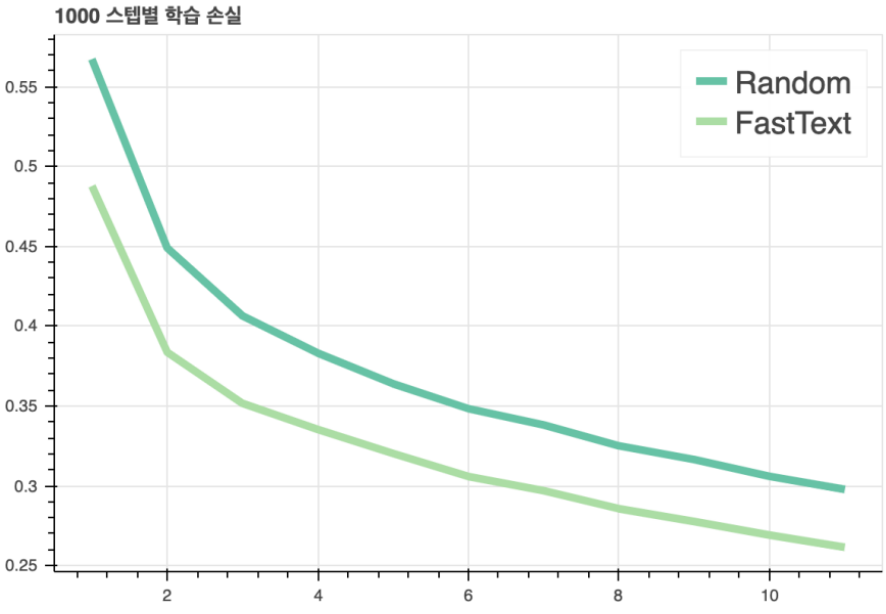
코드 6-10 을 다음으로 교체

```
token_ids.extend([len(self.vocab) - 1] * (max_token_length - tokens_length))
>
token_ids.extend([len(self.vocab) + 1] * (max_token_length - tokens_length))
```

P. 263 (1 쇄 p. 263/ 2 쇄 p. 263/ 3 쇄 p. 265)

그림 6-3 을 다음으로 대체





P. 265 (1 쇄 p. 265/ 2 쇄 p. 265/ 3 쇄 p. 267)

코드 6-18

attention_score = tf.nn.softmax(tf.contrib.layers.fully_connected(inputs=H, num_outputs=1, activation_fn=None))

>

attention_score = tf.nn.softmax(tf.contrib.layers.fully_connected(inputs=H, num_outputs=1, activation_fn=None), axis=1)

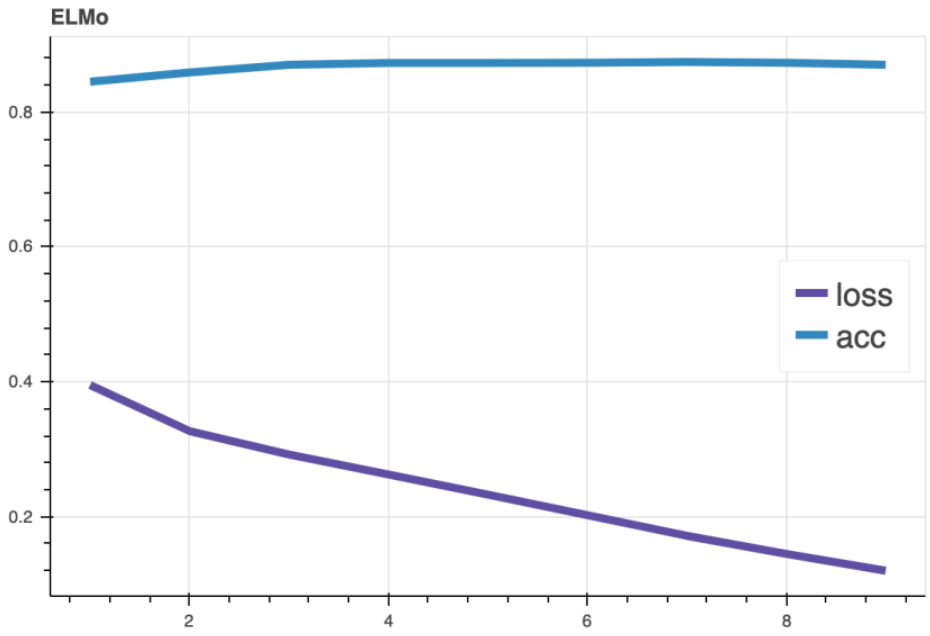
P. 270 (1 쇄 p. 270/ 2 쇄 p. 270/ 3 쇄 p. 272)

ELMo 모델의 베스트 스코어는 100 점 만점에 87.42 점을 기록했다.

>

ELMo 모델의 베스트 스코어는 100 점 만점에 87.44 점을 기록했다.

그림 6-5 를 다음으로 대체



P. 273 (1 쇄 p. 273/ 2 쇄 p. 273/ 3 쇄 p. 275)

BERT 모델에 문장을 입력해 스페셜 토큰 [CLS]에 해당하는 벡터를 추출한다. 이 벡터를 1 개 층의 풀 커넥티드 레이어(Fully-Connected Layer)를 적용한 뒤 소프트맥스를 취해 2 차원(긍정, 부정)의 확률 벡터로 변환한다.

>

BERT 모델에 문장을 입력해 스페셜 토큰 [CLS]에 해당하는 벡터를 추출한다. 트랜스포머 블록은 문장 내 모든 단어 쌍 간 관계를 고려하기 때문에 [CLS] 벡터에는 문장 전체의 의미가 녹아 있다. 이 벡터를 1 개 층의 풀 커넥티드 레이어(Fully-Connected Layer)를 적용한 뒤 소프트맥스를 취해 2 차원(긍정, 부정)의 확률 벡터로 변환한다.

P. 277 (1 쇄 p. 277/ 2 쇄 p. 277/ 3 쇄 p. 279)

우선 워드피스 토크나이저로 형태소 분석한 배치 문장들 앞뒤에 문장 시작을 알리는 스페셜 토큰 [CLS]와 문장 종료를 뜻하는 [SEP]를 붙이고, 이렇게 붙인 결과가 사용자가 정한 max_seq_length 보다 길면 그 길이를 넘어서는 토큰들을 삭제해서 전체 길이를 맞춘 것이 token_seqeunce 다.

>

우선 배치 문장들을 워드피스 토크나이저로 형태소 분석한 뒤 사용자가 정한 길이(max_seq_length - 2)로 자른다. 이후 문장 앞뒤에 문장 시작을 알리는 스페셜 토큰 [CLS]와 문장 종료를 뜻하는 [SEP]를 붙인다.

P. 279 (1 쇄 p. 279/ 2 쇄 p. 279/ 3 쇄 p. 281)

```
sess = tf.Session()
tf.train.Saver(restore_vars).restore(sess, self.pretrain_model_fname)
saver = tf.train.Saver(max_to_keep=1)
sess.run(tf.global_variables_initializer())
>
sess = tf.Session()
sess.run(tf.global_variables_initializer())
tf.train.Saver(restore_vars).restore(sess, self.pretrain_model_fname)
saver = tf.train.Saver(max_to_keep=1)
```

P. 282 (1 쇄 p. 282/ 2 쇄 p. 282/ 3 쇄 p. 284)

이를 t-SNE 로 2 차원을 줄이고 Bokeh 로 시각화한다.
>
이를 t-SNE 로 2 차원으로 줄이고 Bokeh 로 시각화한다.

P. 303 (1 쇄 p. 303/ 2 쇄 p. 303/ 3 쇄 p. 305)

수식 33 에서
a_k a_k^T = 1 > a_k^Ta_k=1
v_k v_k^T = 1 > v_k^Tv_k=1

P. 315 (1 쇄 p.315 / 2 쇄 p.315 / 3 쇄 p.317)

DAG 는 깊이 우선 탐색(Breath First Search) 방식으로 계산한다. 깊이 우선 탐색이란 더 이상 나아갈 엣지가 없을 때까지 차례대로 깊이 탐색하는 그래프 순회(graph traverse) 기법이다.
>
DAG 는 너비 우선 탐색(Breath First Search) 방식으로 계산한다. 너비 우선 탐색이란 시작 노드를 방문한 후 이 노드에 인접한 노드를 우선 탐색하는 그래프 순회(graph traverse) 기법이다.