클라우드 컴퓨팅과 AI서비스
(14주차;15주차)

융합학과 권오영

oykwon@koreatech.ac.kr

학습내용

- ❖ 프로젝트 안내
- ❖ 영화리뷰

기말 프로젝트

프로젝트(기말과제)

❖ 인공지능을 활용한 서비스 제안 및 구현

1. 제안서
   - 2페이지내외 자유형식 (14주차 개인별 발표; 컨설팅)
     (필기숫자인식, 공구의 모양인식 등; 여러분 전공과 연관된 서비스를 자유롭게 정의해보세요.)
2. 서비스 구현보고서
   - 구현에 사용한 알고리즘과 제안한 서비스와 최종 구현물 사이에 발생한 차이점 위주로 5페이지이내로 작성
     * 데이터는 가능하면 공개된 데이터 활용하고, 아니면 어떻게 데이터를 만들었는지 기술
     * 클라우드시스템이 카메라등 외부 입력이 원할하지 않으므로 입력데이터는 외부에서 생성해서 클라우드로 업로드해도 됨
     * 컴퓨터공학부학생은 반드시 streamlit을 이용
3. 소스코드

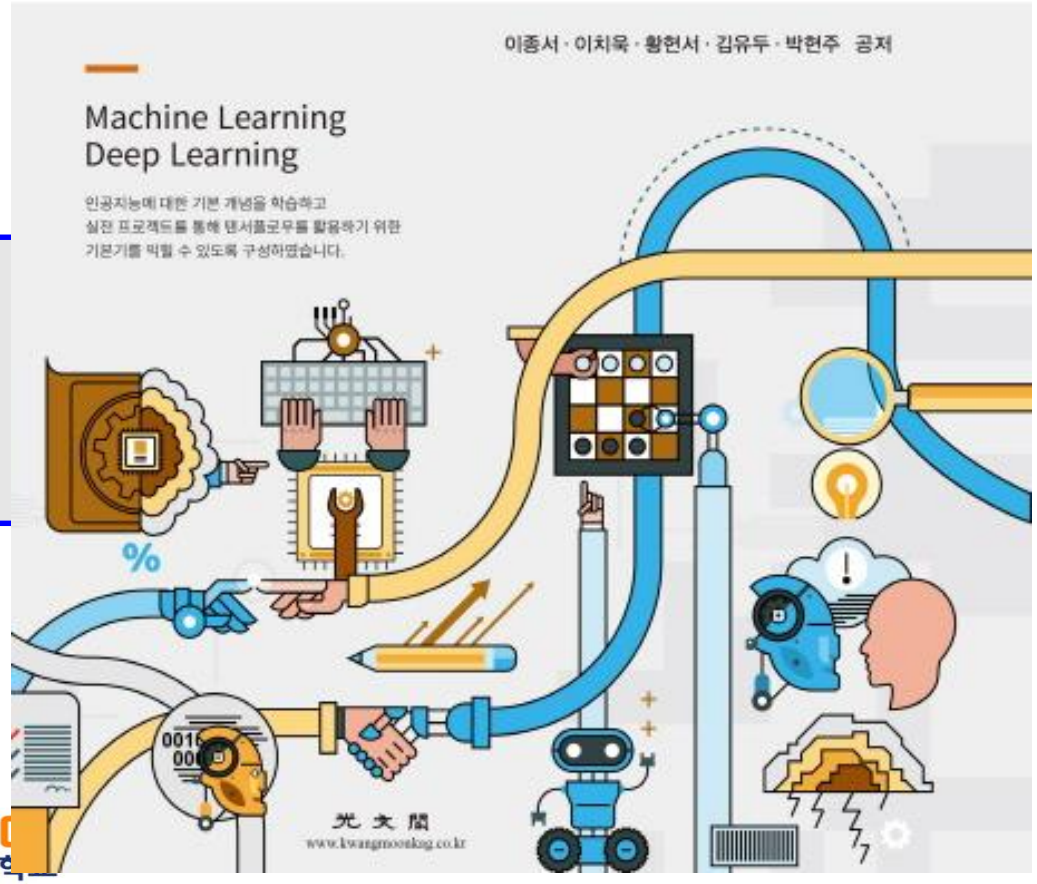
❖ 개인 제출 3분 동영상 형태로 제출 (3분 초과시 감점)

❖ 제출마감일: 12월 19일 15:00

제안서 컨설팅

❖ 개인별 프로젝트 방향 발표

영화리뷰 프로젝트


인공지능을 위한
텐서플로우 TensorFlow
애플리케이션 프로그래밍
이종서 · 이치욱 · 황현서 · 김유두 · 박현주 공저
Machine Learning
Deep Learning
인공지능에 대한 기본 개념을 학습하고
실전 프로젝트를 통해 텐서플로우를 활용하기 위한
기본기를 익힐 수 있도록 구성하였습니다.
光文閣
www.kwangmoonkag.co.kr

프로젝트

- CNN, LSTM을 활용하여 한국어 영화 리뷰 감성 분석 (긍정/부정)

- 영화리뷰 데이터를 가져오는 코드를 직접 수행해보자.
  $ python preprocessing.py

- Package가 없어서 오류가 발생하는데 --- 해결책은?

- 시간이 많이 소요되어서 수행되고 있는지를 어떻게 검증할까?

CNN, LSTM을 활용하여 한국어 영화 리뷰 감성 분석 (긍정/부정)

1. 데이터 수집 (네이버의 영화리뷰 수집)
2. 데이터 전처리 (word2vec에 활용가능한 행태로 처리)
3. CNN/LSTM 모델 구현 및 학습
4. 예측활용

github.com/AI-ML-DL/AI-ML-DL 다운로드 후에 아래 명령으로 cmd환 경에서 수행

```
$docker run –u $(id –u):$(id –g) -it -p 8888:8888 –rm –v ~/mynotebooks:/tf  tensorflow/tensorflow:1.15.2-jupyter bash
```

데이터 수집

❖ Naver Sentimental Movie Corpus 란?

- 한국어로 된 네이버 영화 리뷰를 웹 스크래핑(Web Scraping)한 데이터
- GitHub 주소 - https://github.com/e9t/nsmc
- 20만 개의 리뷰를 15만 개의 트레이닝 데이터와 5만 개의 테스트 데이터로 구성
- 데이터 특성
  - ✓ 모든 리뷰는 140자 미만으로 구성
  - ✓ 긍정/부정은 동일한 비율로 샘플링
  - ✓ 긍정 리뷰는 평점이 9점 이상으로 구성
  - ✓ 부정 리뷰는 평점이 4점 이하로 구성
- 데이터 샘플
  - ✓ id : 네이버에서 제공하는 리뷰 ID값
  - ✓ document : 실제 리뷰 내용
  - ✓ label : 리뷰의 감정 분류
    (0 : 부정, 1 : 긍정)
  - ✓ 각 컬럼 탭으로 구분

```
$ cat ratings_train.txt | head -n 6
id document label
9976970 아 더빙.. 진짜 짜증나네요 목소리 0
3819312 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나 1
```

데이터 수집

❖ Naver Sentimental Movie Corpus 데이터 수집 구현

- GitHub에 존재하는 ratings_train.txt와 ratings_test.txt 다운로드
- 데이터 다운로드 주소
  - ✓ GitHub에서 파일의 이름을 클릭하여 이동한 페이지의 "View Raw"링크를 클릭하여 확인
  - ✓ HTTP Method 중 GET Method를 사용
  - ✓ 응답의 상태 코드 200으로 정상 호출 확인
  - ✓ 응답 온 바이너리 데이터를 파일로 저장

```
8  def nsmc_data_download(file_list) :
9      # 데이터 다운로드 주소
10     nsmc_url = 'https://raw.githubusercontent.com/e9t/nsmc/master/'
11     source_dir ='./data/'
12
13     # 데이터 다운로드 폴더 생성
14     if not(os.path.isdir(source_dir)) :
15         os.makedirs(os.path.join(source_dir))
16
17     # 바이너리 데이터를 파일로 쓰기
18     for file in file_list :
19         response = requests.get(nsmc_url + file)
20         print('file name : ' + file)
21         print('status code : ' + str(response.status_code))
22         with open(source_dir + file,'wb') as f:
23             # 바이너리 형태로 데이터 추출
24             f.write(response.content)
25         f.close()
```

데이터 전처리

❖ 네이버 맞춤법 검사기 란?

- 비격식(Informal) 문장인 영화 리뷰의 맞춤법과 띄어쓰기를 교정함으로써 가독성을 높임
- 네이버 포털(https://www.naver.com) 화면에서 "네이버 맞춤법 검사기"로 검색
  - ✓ 500자 이내의 문장을 "맞춤법, 표준어 의심, 띄어쓰기, 통계적 교정"에 대한 교정 가능

KOREATECH 한국기술교육대학교

데이터 전처리

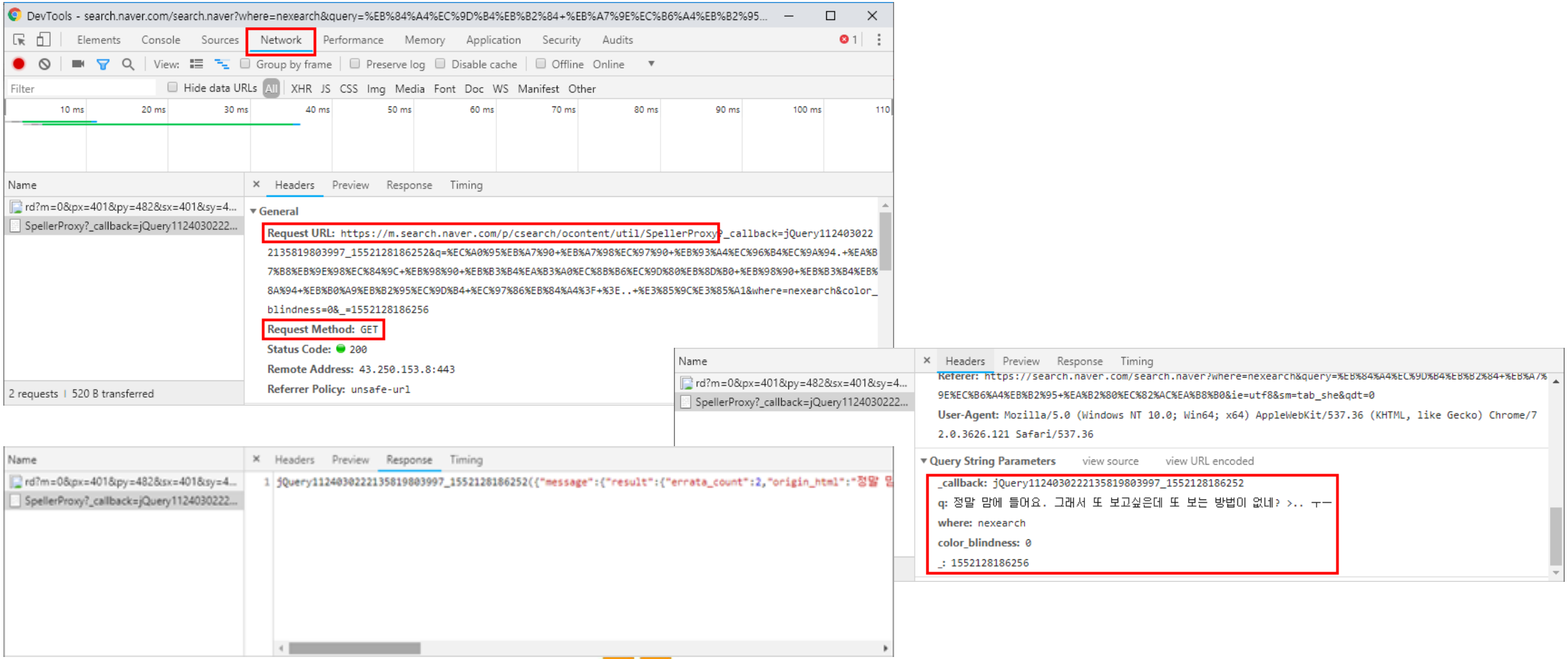
❖ 네이버 맞춤법 검사기 코드 구현

- 네이버 맞춤법 검사기의 주소와 요청 파라미터, 응답 결과를 가져오는 방법(Chrome)
  - ✓ 네이버 맞춤법 화면 이동
  - ✓ F12 개발자 도구 생성
  - ✓ 텍스트 입력 후 검사하기 버튼 클릭
  - ✓ 개발자도구의 Network 탭에서의 왼쪽 Name
  - ✓ Request URL(? 앞까지), Request Method 확
  - ✓ Query String Parameters 확인
  - ✓ Response 탭에서 응답값 확인

```
27  def naver_spell_cheker(input) :
28      source_dir ='./data/'
29      # 네이버 맞춤법 검사기 주소
30      spell_checker_url = 'https://m.search.naver.com/p/csearch/ocontent/util/SpellerProxy'
31
32      def spell_cheker(object) :
33          # request parameter 셋팅
34          req_params = {'_callback': 'SpellChecker', 'q': object, 'color_blindness': 0}
35          while True :
36              response = requests.get(spell_checker_url, params=req_params)
37              status  = response.status_code
38
39              # 응답코드가 200일 때까지 반복
40              if status == 200 :
41                  # 텍스트 형태로 데이터 추출
42                  response = response.text
43                  break
44
45          # json 포멧으로 변경하기 위한 불필요 문자 제거
46          response = response.replace(req_params.get('_callback')+'(', '')
47          response = response.replace(');', '')
48
49          data = json.loads(response)
50          # json 포멧에서 필요 결과 값만 가져오기
51          object = data['message']['result']['notag_html']
52          object = html.unescape(object)
53
54          return object
```

데이터 전처리

❖ 네이버 맞춤법 검사기의 주소와 요청 파라미터, 응답 결과를 가져오는 방법(Chrome)



KOREATECH 한국기술교육대학교

데이터 전처리

❖ 네이버 맞춤법 검사기 코드 구현

○ 응답 값에 대한 샘플

✓ message > result > notag_html 사용

✓ HTML로 escape되어 있는 문자들에 대해 unescape 진행

▪ 많은 문장 수행 시 시간이 오래 소요

```
SpellChecker({
  "message":{
    "result":{
      "errata_count":2,
      "origin_html":"정말 맘에 들어요. 그래서 또 <span class='result_underline'>보고싶은데</span>
       또 보는 방법이 없네? &gt;.. <span class='result_underline'>ㅜㅡ</span>",
      "html":"정말 맘에 들어요. 그래서 또 <em class='green_text'>보고 싶은데</em> 또 보는 방법이 없네?
      &gt;.. <em class='violet_text'>ㅜㅡ</em>",
      "notag_html":"정말 맘에 들어요. 그래서 또 보고 싶은데 또 보는 방법이 없네? &gt;.. ㅜㅡ"
    }
  }
});
```

데이터 전처리

❖ 자연어처리란?

- 사람들이 사용하는 언어에서 의미 있는 정보를 분석하고 추출하여 컴퓨터가 처리
- 음성 인식, 정보 검색, Q&A 시스템, 문서 분류, ChatBot 등
- python의 오픈 소스 라이브러리인 NLTK(Neural Language Toolkit)를 사용
  - ✓ OS에 관련 없이 설치가 가능하지만 한국어에 대한 지원이 지원되지 않음

❖ KoNLPy(코엔엘파이) 란?

- 한국어 자연어 처리를 위해 만들어진 파이썬 오픈 소스 패키지
  - ✓ 어떤 대상 어절을 최소 단위인 "형태소"(단어 자체 또는 단어보다 작은 단위)로 분석
  - ✓ 분석된 결과에 대해 품사를 부착하는 기능
- 형태소 분석기로는 Hannanum, Kkma, Komoran, Mecab, Okt가 제공
  (Okt는 v0.5.0 이전에는 Twitter 클래스로 제공)
  - ✓ 윈도우 환경에서는 Mecab을 지원하지 않음
- 각 형태소 분석기별 비교 - http://konlpy.org/ko/latest/morph/

데이터 전처리

❖ KoNLPy 중 Okt(Open Korean Text) 패키지 구현

- 영화 리뷰에 대해 맞춤법 검사기를 통해 교정된 문장을 형태소 분석 및 품사를 부착
- Okt 제공 함수
  - ✓ Okt().morphs(phrase, norm=False, stem=False)
    - 텍스트를 형태소 단위로 분리. 정규화, 어간 추출
  - ✓ Okt().phrase(phrase)
    - 텍스트에서 어절을 추출
  - ✓ Okt().nouns(phrase)
    - 텍스트에서 명사를 추출
  - ✓ Okt().pos(phrase, norm=False, stem=False, join=False)
    - morphs + 품사
    - join 사용 시 '형태소/품사' 표시

```
83      def pos_tagging(object) :
84          # 형태소 분석 및 품사 태깅(정규화, 어간추출, 품사합치기)
85          pos = Okt().pos(object, norm=True, stem=True, join=True)
86          # 명사 추출
87          noun = Okt().nouns(object)
88
89          return pos, noun
```

데이터 전처리

❖ KoNLPy 중 Okt(Open Korean Text) 패키지 구현

- 자주 사용하는 단어에 대한 시각화를 위해 WordCloud 사용
  - ✓ generate_from_text 함수를 사용
    - 텍스트로 부터 WordCloud 생성
    - 단어와 빈도, 불용어 제거
  - ✓ 빈도가 많은 단어는 크게 표시
- Counter 패키지를 통한 단어의 빈도 계산(상위 20개)
  - ✓ most_common 함수 사용

[('영화', 54418), ('이', 11734), ('정말', 10927), ('것', 10211), ('거', 9284), ('안', 8966), ('진짜', 8483), ('점', 8343), ('보고', 6865), ('연기', 6823), ('최고', 6341), ('평점', 6332), ('수', 6190), ('내', 5644), ('왜', 5602), ('말', 5447), ('스토리', 5377), ('생각', 5304), ('드라마', 5112), ('사람', 4969)]

데이터 전처리

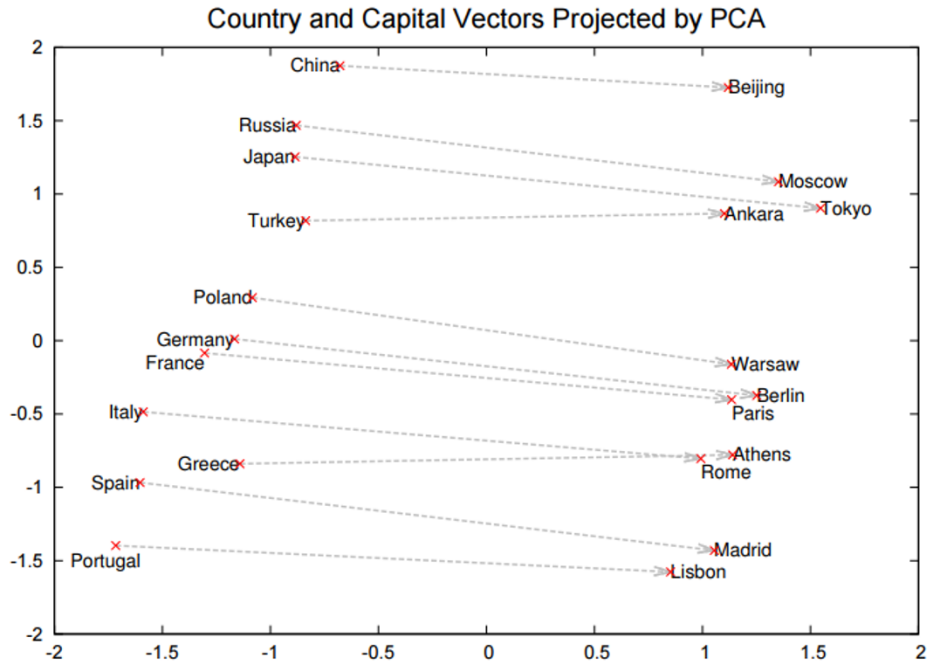
❖ WordCloud 제공 함수

- fit_words(frequencies)
  - ✓ generate_from_frequencies 대한 alias로 단어와 빈도를 통해 wordcloud를 생성
- generate(text)
  - ✓ generate_from_text의 alias로 텍스트를 통해 wordcloud를 생성
  - ✓ process_text 함수와 generate_from_frequencies 함수를 호출
- generate_from_frequencies(frequencies[, …])
  - ✓ 단어와 빈도를 통해 wordcloud를 생성
- generate_from_text(text)
  - ✓ 텍스트를 통해 wordcloud를 생성
  - ✓ process_text 함수와 generate_from_frequencies 함수를 호출
- process_text(text)
  - ✓ 텍스트를 단어로 분리하고 불용어를 제거합니다.
- 파라미터 사용 예시
  - ✓ text = '텍스트 예제입니다 해당 형태로 작성이 필요합니다'
  - ✓ frequencies = {'단어':5, '빈도수':3, '표시':1}

데이터 전처리

❖ 단어 임베딩(Word Embedding)
- 자연어 처리에서 어휘의 단어를 컴퓨터가 처리할 수 있는 실수의 벡터로 변경
- 구문 분석이나 감정 분석에 성능 향상

❖ Word2Vec
- 기존의 one-hot vector 방식의 단어 표현은 단어 간 유사도를 표현할 수 없다는 단점
- 여러 구글 엔지니어(2013년 Tomas Mikolov 외)에 의해 개발한 Neural Network 기반 알고리즘
  - ✓ "비슷한 위치에 등장하는 단어들은 비슷한 의미를 가진다"는 분포 가설(Distributional Hypothesis)
  - ✓ 독일과 베를린이 프랑스와 파리의 같은 방식으로 관련
  - ✓ '파리'-'프랑스'+'이탈리아'는 '로마'
  - ✓ '왕'-'남자'+'여자'는 '여왕'에 가까운 결과



"Distributed Representations of Words and Phrases and their Compositionality", Mikolov, et al. 2013

데이터 전처리

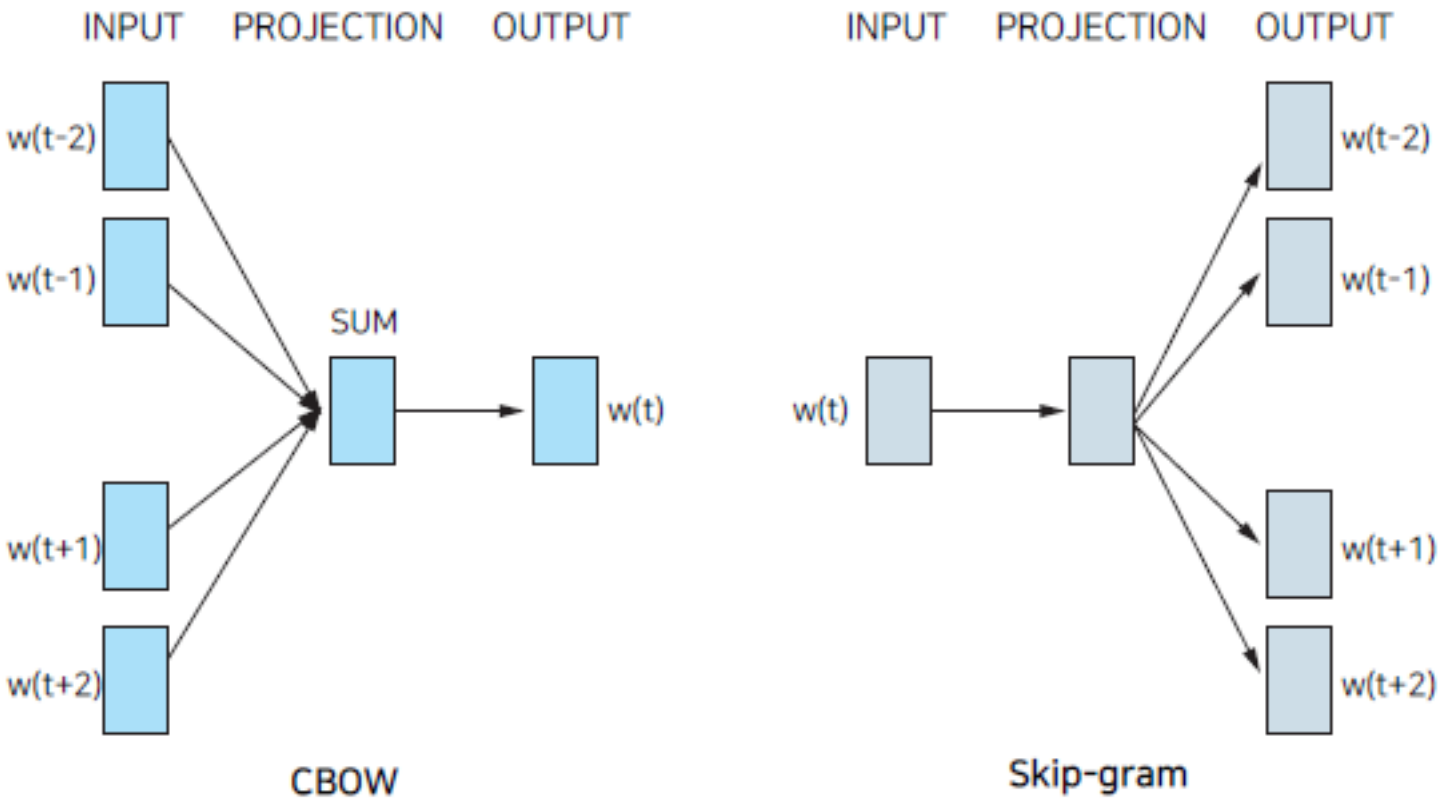
❖ Word2Vec 모델

- CBOW(continuous bag-of-words): 주변에 있는 단어들을 가지고, 중간에 있는 단어들을 예측 (윈도우 2)
  - ✓ ex) ″a barking dog never bites″ - {″a″, ″barking″, ″never″, ″bites″}를 통해 ″dog″를 예측
  - ✓ 타겟 단어(Target Word) - 예측되는 단어
  - ✓ 주변 단어(Context Word) - 예측에 사용되는 단어
  - ✓ 윈도우(Window) - 주변 단어를 앞뒤로 몇 개까지 볼 수 있는지 지정
  - ✓ 슬라이딩 윈도우(Sliding window) - 윈도우를 옆으로 이동하면서 타겟 단어를 바꾸는 것
  - ✓ 크기가 작은 데이터셋에 적합, 속도가 빠른 장점

- Skip-gram: 중간에 있는 단어로 주변 단어들을 예측
  - ✓ ex) “a barking dog never bites”
    윈도우 크기가 2일 때 타겟 단어가 ″dog”라면 그 주변 {“a”, “barking”, “never”, “bites”} 단어를 예측
  - ✓ 타겟 단어의 주변 단어의 배수만큼 학습이 진행, 속도는 느리지만 성능이 더 좋은 결과

데이터 전처리

❖ Word2Vec 모델



"Efficient Estimation of Word Representation in Vector Space", Mikolov, et al. 2013

데이터 전처리

❖ Word2Vec 모델

- CBOW(continuous bag-of-words)
  - ✓ 윈도우가 2인 경우

- Skip-gram
  - ✓ 윈도우가 2인 경우

KOREATECH 한국기술교육대학교

데이터 전처리

❖ Word2Vec 구현

- Gensim 패키지 내 model 클래스 사용
- Word2Vec 파라미터
  - ✓ size :단어를 벡터값으로 변환하기 위한 차원 수, 단어의 전체 개수에 따라 유동적으로 변경 필요
  - ✓ window : 문장 내 현재 단어와 예측 단어와의 최대 거리
  - ✓ min_count : 해당 값보다 낮은 빈도수 단어는 무시
  - ✓ workers : 모델을 학습하기 위한 병렬 처리 쓰레드 개수
  - ✓ sg : CBOW=0, Skip-gram=1
  - ✓ iter : 반복 학습 횟수

```
10  def apply_word2vec(file_list) :
11      source_dir ='./data/'
12      # 전체 문장을 담는 리스트 선언
13      total_sentences = list()
14
15      for file in file_list:
16          with open(source_dir + file, 'r', encoding='UTF-8') as f:
17              load_data = [line.split('\t') for line in f.read().splitlines()]
18              for data in load_data :
19                  total_sentences.append(data[0].split())
20
21      # word2vec로 단어 벡터로 변경 및 모델 저장
22      model = models.Word2Vec(total_sentences, min_count=3, window=5, sg=1, size=100, workers=4, iter=50)
23      model.save(source_dir + '3_word2vec_nsmc.w2v')
24      model.wv.save_word2vec_format(source_dir + '3_word2vec_nsmc_format.w2v', binary=False)
```

데이터 전처리

❖ Word2Vec 구현

- save_word2vec_format 함수를 통해 단어가 벡터로 변경된 내용에 대해 확인 가능
  - ✓ 첫 번째 라인의 두 숫자 : 단어의 전체 개수와 벡터 차원 수
  - ✓ 두 번째 라인부터 단어에 대한 벡터 표현

```
22615 100
영화/Noun -0.32818502 0.27860388 0.1523643 0.06831967 0.19623944 -0.07209147
-0.18959798 -0.14979233 0.4294706 -0.38620764 -0.36520967 -0.005111015
-0.16903515 -0.39880487 0.19440751 -0.21463037 -0.0055839033 0.0191054
-0.13867551 -0.36297414 0.21187727 -0.12709628 -0.03174775 -0.041058134 0.14193992
-0.4235093 0.20149146 0.031723544 -0.09822699 0.19908044 -0.08718104 -0.046228327
0.09436537 0.13143788 -0.033787344 -0.21516575 0.11618206 0.27091342 0.05722208
-0.46929833 -0.074162 -0.1251334 0.15102917 -0.3119958 -0.02536161 -0.31003794
0.101542465 0.29173 -0.15800369 0.0266653 -0.08451466 0.07947255 -0.12946346
0.077833004 -0.18809474 0.022096505 0.23722965 0.019673629 -0.053121354 0.06628938
… 중략
```

데이터 전처리

❖ Word2Vec 모델 테스트

- "배우, 엄마, 여자, 남자"의 단어를 사용 - "배우/Noun, 엄마/Noun, 여자/Noun, 남자/Noun"

```
34      # word2vec 모델 로드
35      model = models.Word2Vec.load(source_dir + w2v_name)
36
37      # 품사 태깅 된 데이터 추출 및 리스트 저장
38      data_list = list()
39      data1 = pre.konlpy_pos_tag('배우')
40      data_list.append(data1)
41      data2 = pre.konlpy_pos_tag('엄마')
42      data_list.append(data2)
43      data3 = pre.konlpy_pos_tag('여자')
44      data_list.append(data3)
45      data4 = pre.konlpy_pos_tag('남자')
46      data_list.append(data4)
47
48      # 모델에 적용하여 결과 출력
49      # model.doesnt_match, model.most_similar의 method는 4.0.0 버전에서 deprecated
50      print(model[data1])
51      print(model.wv.doesnt_match(data_list))
52      print(model.wv.most_similar(positive=[data1], topn=10))
53      print(model.wv.most_similar(positive=[data2, data4], negative=[data3], topn=1))
54      print(model.wv.similarity(data1, data2))
55      print(model.wv.similarity(data1, data3))
```

데이터 전처리

❖ Word2Vec 모델 테스트

- "배우, 엄마, 여자, 남자"의 단어를 사용 - "배우/Noun, 엄마/Noun, 여자/Noun, 남자/Noun"
  - ✓ model['배우/Noun'] : 100차원 벡터 표현
  - ✓ doesnt_match 함수 : 유사도가 없는 결과 표현
    - 결과 : "배우/Noun"
  - ✓ most_similar 함수 : 유사도가 높은 상위 N개의 단어 추출
    - positive :단어와 긍정적인 단어의 리스트
    - negative : 단어와 부정적인 단어 리스트
    - topn : 유사도가 높은 상위 n개의 단어를 반환
    - similarity : 두 단어 사이 유사도
      - » "배우/Noun"과 유사도가 높은 10개의 단어
        - » 결과: [('연기자/Noun', 0.7866206169128418), ('여배우/Noun', 0.7009295225143433), '조연/Noun', 0.6413561701774597), ('영화배우/Noun', 0.6213721632957458), ('열연/Noun', 0.6110374927520752)]
      - » "엄마/Noun"와 "남자/Noun"에 긍정적이고, "여자/Noun"에 부정적인 최상위 단어
        - » 결과 : [('아빠/Noun', 0.7548638582229614)
      - » "배우/Noun"와 "여자/Noun"의 유사도
        - » 결과 : 0.39090595

데이터 전처리

❖ Word2Vec 모델 시각화

- t-SNE(t-Stochastic Neighbor Embedding) 란?
  - ✓ 고차원 벡터 차원을 축소하여 표시
  - ✓ 차원 축소 시 축의 위치 변경에 따라 다른 모양으로 변환
  - ✓ 군집성은 유지
  - ✓ n_component : 축소할 차원 수
- 한글에 대한 시각화
  - ✓ 폰트 검색 - matplotlib 패키지
    - font_manager.get_fontconfig_fonts() 함수

```
['/usr/share/fonts/truetype/nanum/NanumGothicBold.ttf', '/usr/share/fonts/truetype/nanum/NanumMyeongjo.ttf', '/usr/share/fonts/truetype/nanum/NanumMyeongjoBold.ttf', '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf', '/usr/share/fonts/truetype/nanum/NanumGothic_Coding.ttf',
… 중략
]
```

  - ✓ 폰트 설정 - matplotlib 패키지
    - rc() 함수

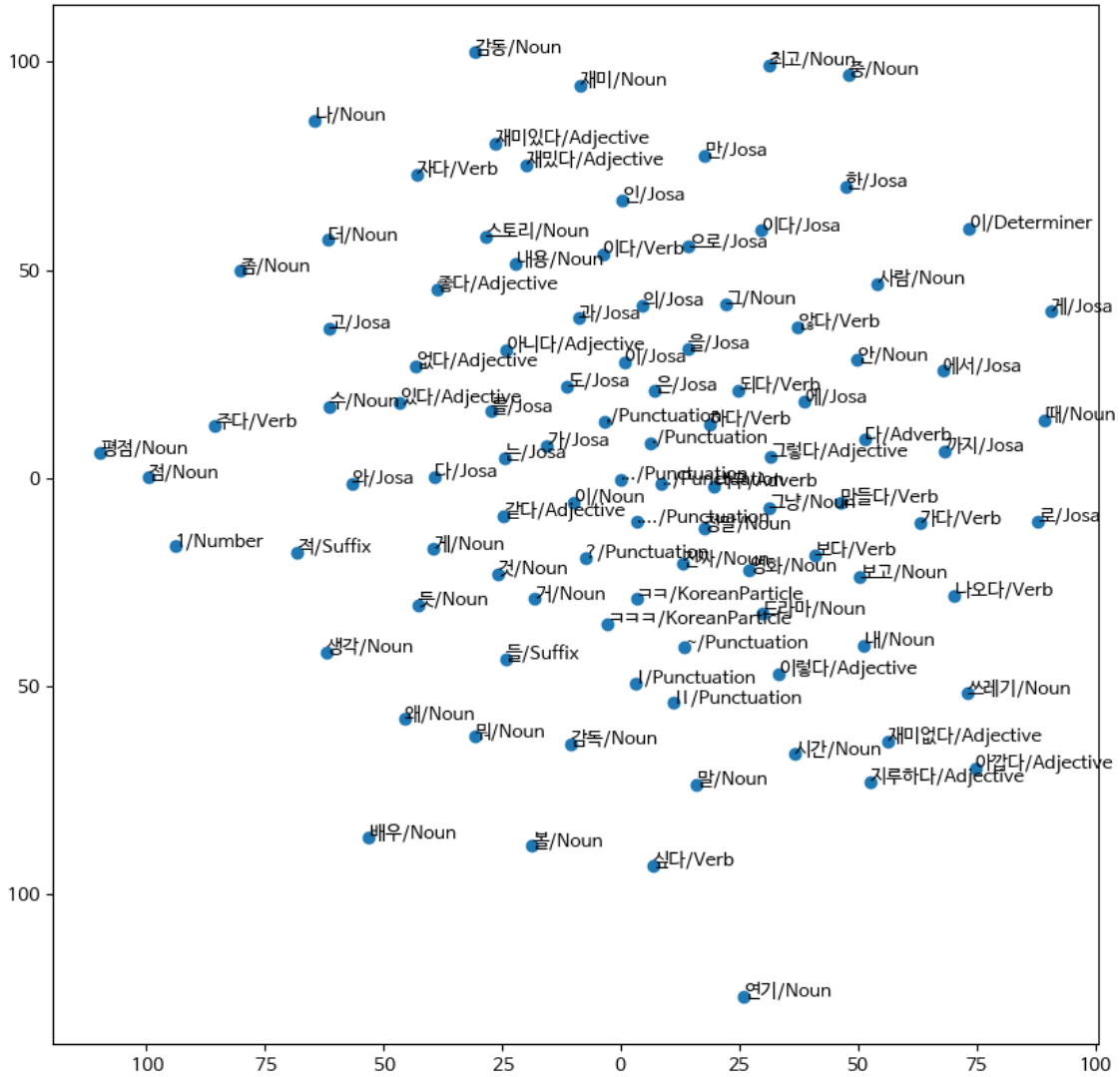
```
63  # 단어 리스트 중 가장 많이 사용된 100개 단어 추출
64  counter = Counter(total_word_list).most_common(100)
65  word_list = [word[0] for word in counter]
66  print(word_list)
67
68  # 설정 가능한 폰트 리스트 출력
69  font_list = font_manager.get_fontconfig_fonts()
70  print([font for font in font_list if 'nanum' in font])
71
72  # 폰트 설정
73  rc('font', family=font_manager.FontProperties(fname=font_name).get_name())
74
75  # 단어에 대한 벡터 리스트
76  vector_list = model[word_list]
77
78  # 2차원으로 차원 축소
79  transformed = TSNE(n_components=2).fit_transform(vector_list)
80  print(transformed)
81
82  # 2차원의 데이터를 x, y 축으로 저장
83  x_plot = transformed[:, 0]
84  y_plot = transformed[:, 1]
85
86  # 이미지의 사이즈 셋팅
87  pyplot.figure(figsize=(10, 10))
88
89  # x, y 축을 점 및 텍스트 표시
90  pyplot.scatter(x_plot, y_plot)
91  for i in range(len(x_plot)):
92      pyplot.annotate(word_list[i], xy=(x_plot[i], y_plot[i]))
93
94  # 이미지로 저장
95  pyplot.savefig(source_dir + fig_file)
```

KOREATECH 한국기술교육대학교

데이터 전처리

❖ Word2Vec 모델 시각화

- t-SNE 결과
  - ✓ 의미가 유사한 단어들이 거리가 가깝게 표시
  - ✓ "재미있다, 재밌다", "감동, 재미"의 단어가 거리가 가깝게 표현
  - ✓ "재미없다, 아깝다, 지루하다"의 단어들도 가깝게 표시

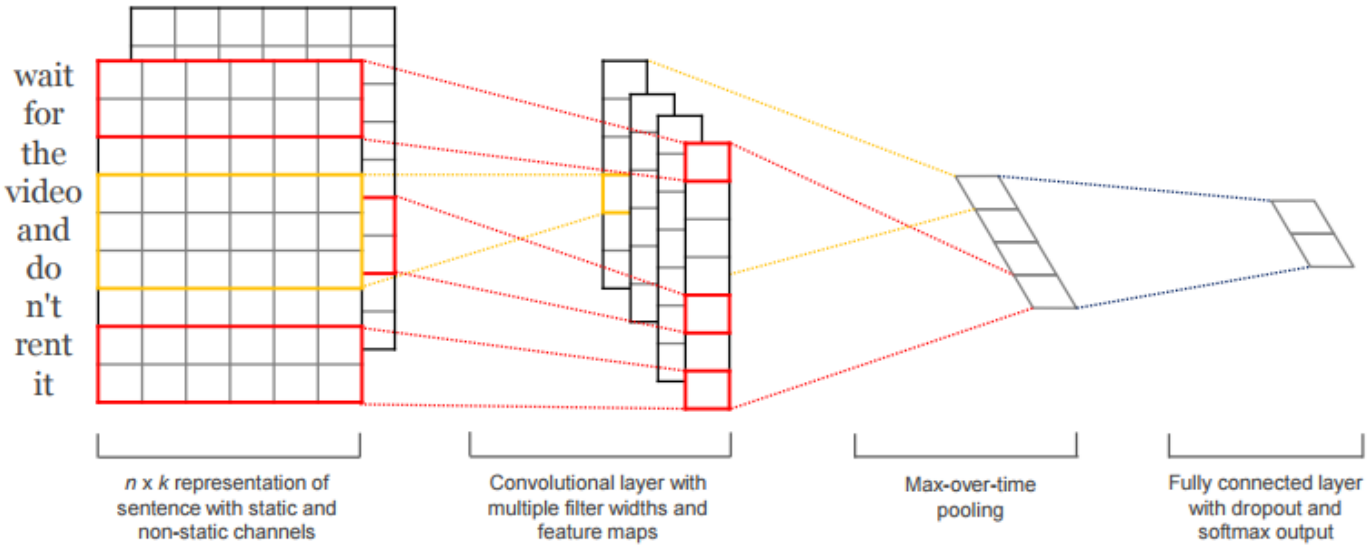


KOREA TECH 한국기술교육대학교

합성곱 신경망(CNN) 구현

❖ 자연어 처리에서의 합성곱 신경망 구조

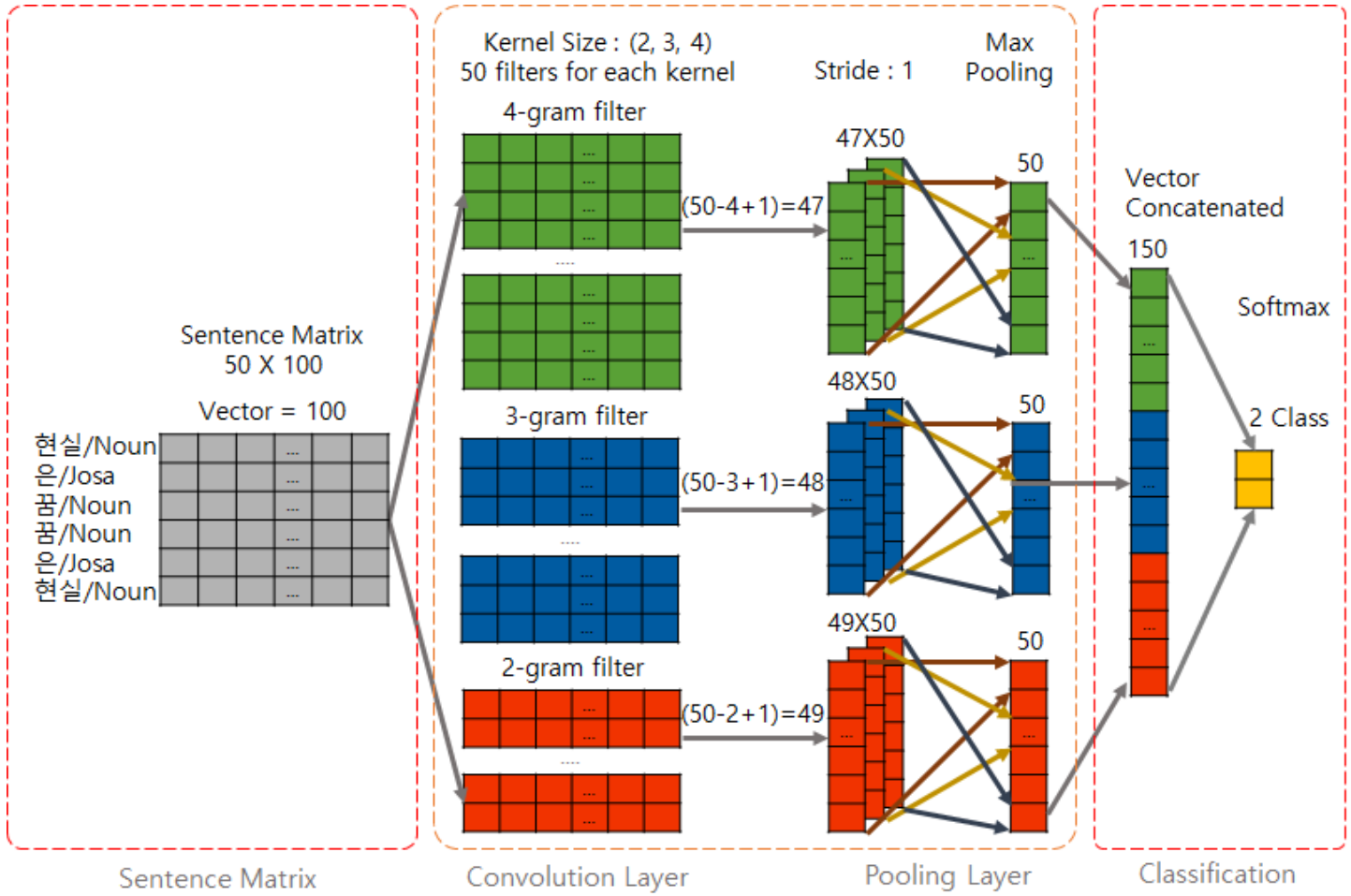
- 2014년 김윤 논문 - Convolutional Neural Network for Sentence Classification
  - ✓ 단어들을 벡터화, 여러 필터 크기를 사용하여 합성곱 및 특성 맵을 구성
  - ✓ 최댓값 풀링, 드롭아웃과 Softmax를 통해 결괏값을 분류



"Convolutional Neural Network for Sentence Classification", Yoon Kim, 2014

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 전반적 구조



KOREATECH 한국기술교육대학교

합성곱 신경망(CNN) 구현

❖ 전처리 단계 - Step 1) 파라미터 설정, Word2Vec 로드

- 파라미터 설정
  - ✓ 단어의 벡터 차원 수 -> embedding_dim : 100
  - ✓ 레이블 차원 수 -> class_sizes : 2
  - ✓ 문장 내 최대 단어 개수 -> max_sentence_length : 50
  - ✓ 합성곱 필터 사이즈 -> filter_sizes : [2 3 4]
  - ✓ 합성곱 특성 맵 개수 -> num_filters : 50
  - ✓ 학습 시 드롭아웃 변수 -> dropout_keep_prob : 0.5
  - ✓ 학습 횟수 -> num_epochs : 20
  - ✓ 패치 사이즈 -> batch_size : 1000
  - ✓ 검증을 위한 조건 -> evaluate_every : 150
  - ✓ 최적화 알고리즘 학습률 -> learn_rate : 0.001
- Word2Vec 로드
  - ✓ Word2Vec.load() 함수를 통해 저장된 모델 로드
  - ✓ 모델에서의 vector_size 셋팅

```
72  def run_cnn(params) :
73      class_sizes = 2
74      max_sentence_length = int(params[1])
75      filter_sizes = np.array(params[2].split(','), dtype=int)
76      num_filters = int(params[3])
77      dropout_keep_prob = float(params[4])
78      num_epochs = int(params[5])
79      batch_size = int(params[6])
80      evaluate_every = int(params[7])
81      learn_rate = float(params[8])
82
83      model = Word2Vec.load(source_dir + w2v_file_name)
84      # word2vec 파일에서의 벡터 차원 수 계산
85      embedding_dim = model.vector_size
```

합성곱 신경망(CNN) 구현

❖ 전처리 단계 - Step 2) 데이터 준비

- 트레이닝, 테스트 데이터 로드
- 데이터 구조 셋팅
  - ✓ data size * word_length * embedding
- 최대 단어 크기 셋팅 및 Word2Vec 변환
- 라벨 셋팅 - One Hot Encoding

```python
def data_setting(w2v_model, embedding_dim, class_sizes, max_word_length):
    # 데이터 불러와서 문장의 총 개수 셋팅
    train_data = load_data(source_dir + file_list[0])
    train_size = len(train_data)
    #print('train_size : ' + str(train_size))

    test_data = load_data(source_dir + file_list[1])
    test_size = len(test_data)
    #print('dev_size : ' + str(dev_size))

    # 데이터 구조 : 전체 문장 x 문장 내 단어 제한 수 x 벡터의 차원
    train_arrays = np.zeros((train_size, max_word_length, embedding_dim))
    test_arrays = np.zeros((test_size, max_word_length, embedding_dim))
    # 정답의 구조 : 전체 문장 x 구분 수(긍정/부정)
    train_labels = np.zeros((train_size, class_sizes))
    test_labels = np.zeros((test_size, class_sizes))

    for train in range(len(train_data)) :
        # 각 문장의 단어를 벡터화 하고 문장 구성
        train_arrays[train] = pre.max_word_length_word2vec(w2v_model, embedding_dim, max_word_length, train_data[train][0])
        # 각 문장이 정답을 one-hot encoding으로 변경
        train_labels[train] = label_value(int(train_data[train][1]), class_sizes)

    for dev in range(len(test_data)) :
        test_arrays[dev] = pre.max_word_length_word2vec(w2v_model, embedding_dim, max_word_length, test_data[dev][0])
        test_labels[dev] = label_value(int(test_data[dev][1]), class_sizes)

    return train_arrays, train_labels, test_arrays, test_labels
```

합성곱 신경망(CNN) 구현

❖ 전처리 단계 - Step 2) 데이터 준비

- 데이터 로드
  - ✓ 문장을 읽어 탭으로 구분

```
14  # 파일을 읽어 각 문장을 탭으로 구분
15  def load_data(txtFilePath):
16      #with open(txtFilePath,'r', encoding='UTF-8') as data_file:
17      with open(txtFilePath,'r') as data_file:
18          return [line.split('\t') for line in data_file.read().splitlines()]
```

- 최대 단어 크기 셋팅 및 Word2Vec 변환
  - ✓ 최대 크기 단어 설정
    - 너무 크게 잡으면 성능 저하
    - 최대 단어 크기를 넘어가는 문장 내 단어 삭제
    - 최대 단어 크기보다 부족한 문장은 제로 패딩
  - ✓ 단어들에 대해 벡터 변환
  - ✓ Word2Vec 모델에 존재하지 않는 벡터 값은 제외

```
132 def max_word_length_word2vec(w2v_model, embedding_dim , max_word_length, word_list):
133     # 문장 내 단어 제한 x 벡터 차원 수
134     data_arrays = np.zeros((max_word_length, embedding_dim))
135
136     # string 문장으로 들어오는 경우 split 처리
137     if type(word_list) is str :
138         word_list = word_list.split()
139
140     # 단어를 벡터로 변경
141     if len(word_list) > 0 :
142         word_length = max_word_length if max_word_length < len(word_list) else len(word_list)
143
144         for i in range(word_length):
145             try :
146                 data_arrays[i] = w2v_model[word_list[i]]
147             except KeyError :
148                 pass
149     return data_arrays
```

합성곱 신경망(CNN) 구현

❖ 전처리 단계 - Step 2) 데이터 준비

- 라벨 값 셋팅
  - ✓ One-Hot Encoding으로 설정
  - ✓ 긍정은 [0.1]
  - ✓ 부정은 [1.0] 으로 설정

```
# 긍/부정에 대한 one-hot encoding
def label_value(code, size):
    code_arrays = np.zeros((size))
    # 부정인 경우 [1, 0]
    if code == 0:
        code_arrays[0] = 1
    # 긍정인 경우 [0, 1]
    elif code == 1:
        code_arrays[1] = 1

    return code_arrays
```

❖ 합성곱 신경망 모델 - Step 1) 모델 생성을 위한 변수 초기화

- input_x : Word2Vec를 통해 문장에서의 각 단어별 벡터 데이터
  - ✓ 합성곱 계층에 들어가는 데이터는 4D 텐서로 표현되기 때문에 expand_dims을 통해 차원을 확장
  - ✓ [batch_size, sequence_length, embedding_size, 1] 형태
- input_y : 긍정/부정 값에 대한 one-hot encoding된 데이터
- dropout_keep_prob : 드롭아웃하지 않고 유지할 노드의 비율

```
class cnn_model(object):
    def __init__(self, sequence_length, num_classes, embedding_size, filter_sizes, num_filters):
        # 학습 데이터가 들어갈 플레이스 홀더 선언
        self.input_x = tf.placeholder(tf.float32, [None, sequence_length, embedding_size], name="input_x")
        self.input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")
        self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")
        self.expanded_input_x = tf.expand_dims(self.input_x, -1)
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 - Step 2) 합성곱/풀링 레이어

- 합성곱 레이어
  - ✓ [filter_size, embedding_size, 1, num_filters] 형태의 필터 와 expanded_input_x 입력값의 합성곱
  - ✓ 가로, 세로 1칸씩 이동하는 스트라이드 설정
  - ✓ 특성 맵(feature map) : sentence_length - filter_size + 1
  - ✓ 특성 맵의 개수인 num_filters 변숫값을 통해
    [batch_size, sentence_length - filter_size + 1, 1, num_filters] 형태의 합성곱 텐서가 생성

```
# 각 필터별 합성곱 레이어 + 풀링 레이어 생성
pooled_outputs = list()
for i, filter_size in enumerate(filter_sizes):
    with tf.name_scope("conv-maxpool-%s" % filter_size):
        # 합성곱 레이어
        filter_shape = [filter_size, embedding_size, 1, num_filters]
        W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W")
        b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
        conv = tf.nn.conv2d(self.expanded_input_x, W, strides=[1, 1, 1, 1], padding="VALID", name="conv")
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")

        # 맥스 풀링 레이어
        pooled = tf.nn.max_pool(h, ksize=[1, sequence_length - filter_size + 1, 1, 1], strides=[1, 1, 1, 1], padding='VALID', name="pool")
        pooled_outputs.append(pooled)
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 - Step 2) 합성곱/풀링 레이어

- 풀링 레이어
  - ✓ 합성곱 레이어와 동일한 [1, sequence_length - filter_size + 1, 1, 1]인 커널
  - ✓ 가로, 세로 1칸씩 이동하는 스트라이드 설정
  - ✓ 최댓값 풀링을 적용하여 [batch_size, 1, 1, num_filters] 형태의 결과가 출력
  - ✓ Padding 옵션
    - "VALID" : 제로 패딩을 하지 않고 스트라이드에 따라 오른쪽의 행, 열 값이 무시
    - "SAME" : 제로 패딩을 사용하여 똑같은 크기의 차원이 리턴
  - ✓ pooled_outputs : filter_sizes에 따른 합성곱 계층 과 풀링 계층의 동일한 연산을 3번 수행
  - ✓ 3개의 풀링 데이터를 합쳐 [batch_size, num_filters_total] 형태의 Fully-Connected Layer 생성

```
# 풀링된 데이터 통합 및 차원 변경
num_filters_total = num_filters * len(filter_sizes)
self.h_pool = tf.concat(pooled_outputs, 3)
self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 - Step 2) 합성곱/풀링 레이어

- 드롭 아웃 및 최종 출력 레이어
  - ✓ 학습 중에는 0.5로 절반을 랜덤으로 비활성성화 검증에서는 1.0으로 비활성화하지 않도록 설정
  - ✓ Softmax를 통해 최종 분류를 수행
  - ✓ 가중치의 초기화 : tf.contrib.layers.xavier_initializer 함수
    - 2010년 Glorot과 Bengio가 발표
    - Xavier는 입력값과 출력값의 난수를 선택하여 입력값의 제곱근으로 나누는 것
      - » W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)

```
31          # 드랍아웃 적용
32          with tf.name_scope("dropout"):
33              self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)
34
35          # Output Layer
36          with tf.name_scope("output"):
37              W = tf.get_variable("W", shape=[num_filters_total, num_classes], initializer=tf.contrib.layers.xavier_initializer())
38              b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b")
39              self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")
40              self.predictions = tf.argmax(self.scores, 1, name="predictions")
41              self.result =  tf.nn.softmax(logits=self.scores, name="result")
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 - Step 3) 비용 / 정확도 계산

비용 함수

✓ 비용 함수 : softmax_cross_entropy_with_logits_v2

– 최신 Tensorflow 버전에서는 tf.nn.softmax_cross_entropy_with_logits가 deprecated될 예정

```
43          # 비용 함수(오차, 손실함수) 선언
44          with tf.name_scope("loss"):
45              # v2 아닌 method는 deprecated 예정
46              self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=self.scores, labels=self.input_y))
47
48          # 정확도 계산
49          with tf.name_scope("accuracy"):
50              correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, axis=1))
51              self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32), name="accuracy")
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 학습 실행

- tf.Graph().as_default()를 통해 그래프를 생성
  - ✓ tf.ConfigProto() : gpu_options.allow_growth 와 같은 GPU를 사용하는 만큼만 증가시키는 옵션들을 설정
  - ✓ 옵션들은 tf.Session(config=sess_config)에 파라미터로 넘겨 사용

```
115     with tf.Graph().as_default():
116         #sess_config = tf.ConfigProto(device_count = {'GPU': 0})
117         sess_config = tf.ConfigProto()
118         sess_config.gpu_options.allow_growth = True
119         sess = tf.Session(config=sess_config)
```

- sess.as_default()를 사용하여 세션 범위 지정
  - ✓ 합성곱 신경망에 사용할 변수 초기화

```
121         with sess.as_default():
122             cnn = cnn_model(
123                 sequence_length=x_train.shape[1],
124                 num_classes=y_train.shape[1],
125                 embedding_size=embedding_dim,
126                 filter_sizes=filter_sizes,
127                 num_filters=num_filters)
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 학습 실행

- 최적화 함수 : AdamOptimizer
  - ✓ 비용 함수의 값이 최소화 되도록 minimize() 함수 사용
    - compute_gradients + apply_gradients 함수 사용과 동일
- 모델과 파라미터를 저장하기 위해 tf.train.Saver() 사용
- 변수들을 초기화하기 위해 tf.global_variables_initializer()를 수행

```
130             # 비용함수의 값이 최소가 되도록 하는 최적화 함수 선언
131             optimizer = tf.train.AdamOptimizer(learn_rate)
132             train_op = optimizer.minimize(cnn.cost, global_step=global_step)
133             #grads_and_vars = optimizer.compute_gradients(cnn.cost)
134             #train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)
135
136             saver = tf.train.Saver()
137             sess.run(tf.global_variables_initializer())
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 학습 실행

- 학습을 위해 배치 트레이닝 방식을 사용
  - ✓ make_batch 함수를 구현하여 배치 사이즈만큼 학습할 수 있도록 리스트를 생성

```
61  def make_batch(list_data, batch_size):
62      num_batches = int(len(list_data)/batch_size)
63      batches = list()
64
65      for i in range(num_batches):
66          start = int(i * batch_size)
67          end = int(start + batch_size)
68          batches.append(list_data[start:end])
69
70      return batches
```

- 배치별 트레이닝, 검증 작업 수행
  - ✓ 1,000건 단위로 배치를 생성
  - ✓ 학습 step 값이 150으로 나눠떨어질 때마다 테스트 데이터를 통해 학습된 모델에 대한 검증

```
168             # 배치별 트레이닝, 검증
169             for epoch in range(num_epochs):
170                 for len_batch in range(len(train_x_batches)):
171                     train_step(train_x_batches[len_batch], train_y_batches[len_batch])
172                     current_step = tf.train.global_step(sess, global_step)
173                     if current_step % evaluate_every == 0:
174                         dev_step(x_dev, y_dev, epoch)
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 학습 실행

- train_step 과 dev_step에서의 feed_dict 값의 차이는 dropout 값
- 각 step 값과 정확도, 비용 값을 시각화를 위해 리스트로 저장
- 학습 결과
  - ✓ 총 20회의 학습
  - ✓ 비용 값은 0.385에서 0.314로 감소
  - ✓ 정확도는 83.1%에서 86.7%로 증가

```
def train_step(x_batch, y_batch):
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: dropout_keep_prob
    }
    _, step, cost, accuracy = sess.run([train_op, global_step, cnn.cost, cnn.accuracy], feed_dict)
    train_x_plot.append(step)
    train_y_accracy.append(accuracy * 100)
    train_y_cost.append(cost)
    #print("Train step {}, cost {:g}, accuracy {:g}".format(step, cost, accuracy))
```

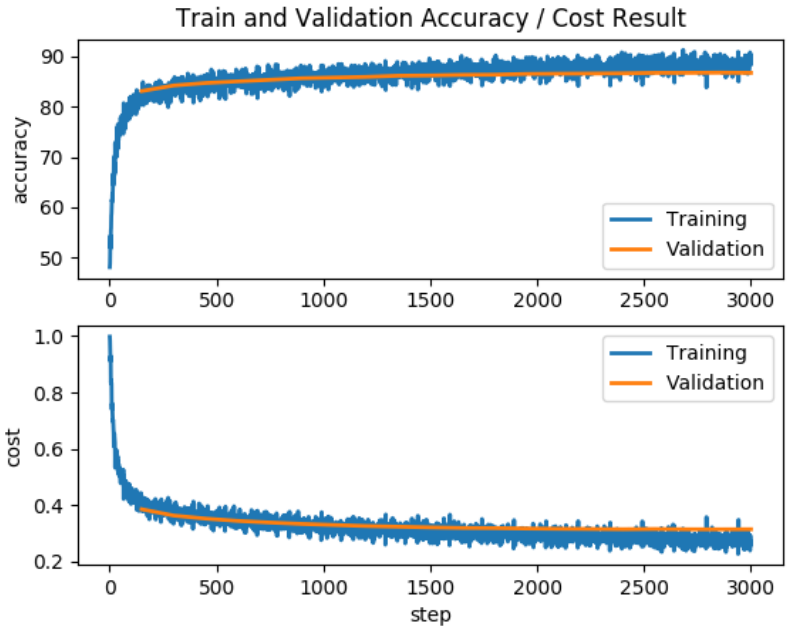
```
def dev_step(x_batch, y_batch, epoch):
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: 1.0
    }
    step, cost, accuracy, dev_pred = sess.run([global_step, cnn.cost, cnn.accuracy, cnn.predictions], feed_dict)
    valid_x_plot.append(step)
    valid_y_accuracy.append(accuracy * 100)
    valid_y_cost.append(cost)
    print("Valid step, epoch {}, step {}, cost {:g}, accuracy {:g}".format((epoch+1), step, cost, accuracy))
```

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 학습 실행

- 학습과 검증에서의 정확도와 비용에 대한 시각화
  - ✓ 이미지 파일로 저장

```
178     # 학습 / 검증에서의 정확도와 비용 시각화
179     plt.subplot(2,1,1)
180     plt.plot(train_x_plot, train_y_accracy, linewidth = 2, label = 'Training')
181     plt.plot(valid_x_plot, valid_y_accuracy, linewidth = 2, label = 'Validation')
182     plt.title("Train and Validation Accuracy / Cost Result")
183     plt.ylabel('accuracy')
184     plt.legend()
185
186     plt.subplot(2,1,2)
187     plt.plot(train_x_plot, train_y_cost, linewidth = 2, label = 'Training')
188     plt.plot(valid_x_plot, valid_y_cost, linewidth = 2, label = 'Validation')
189     plt.xlabel('step')
190     plt.ylabel('cost')
191     plt.legend()
192
193     # 이미지로 저장
194     plt.savefig(source_dir + fig_file_name)
```



KOREATECH 한국기술교육대학교

합성곱 신경망(CNN) 구현

❖ 합성곱 신경망 모델 학습 실행

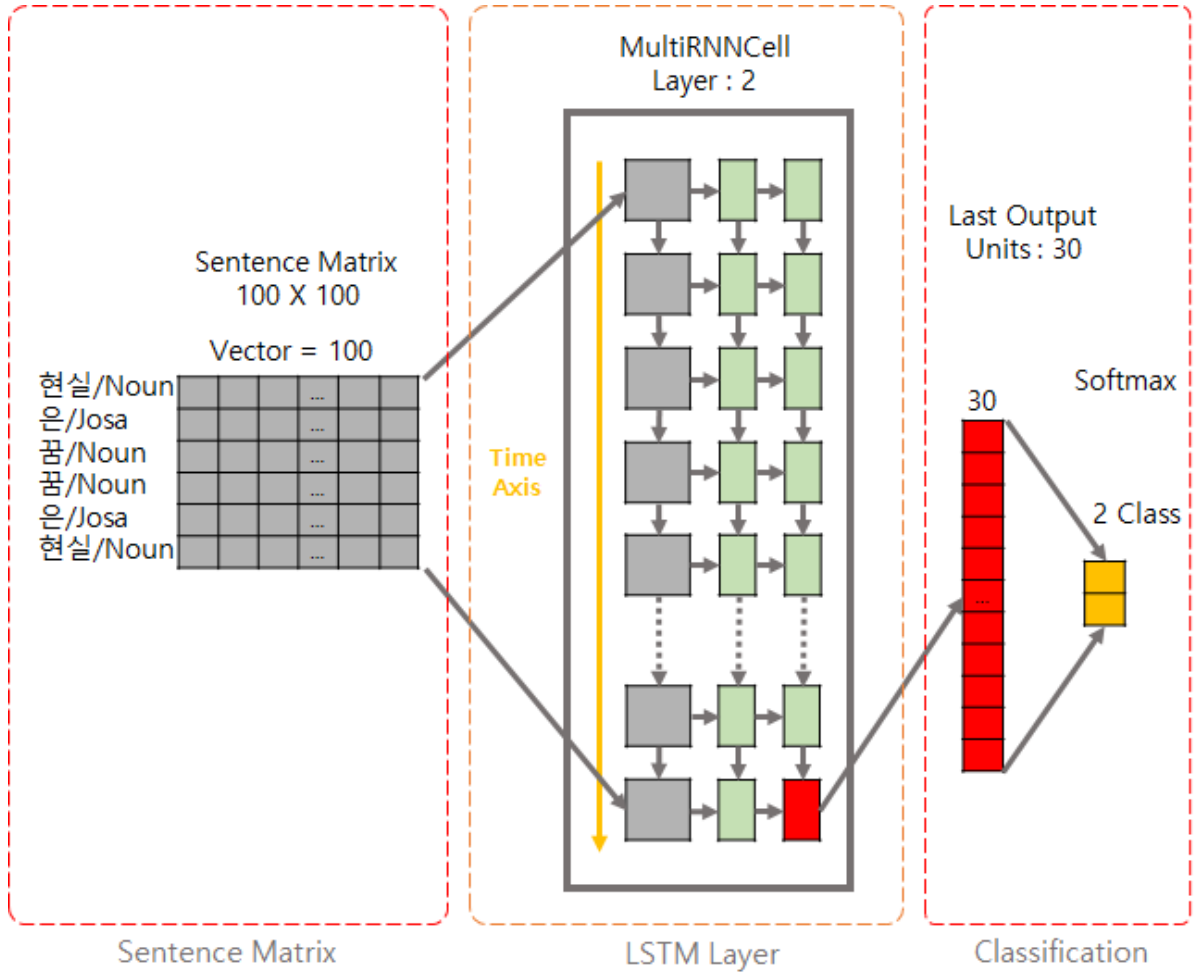
- 학습이 완료된 모델 저장

```
175             # 모델 저장
176             saver.save(sess, "./cnn_model/model.ckpt")
```

checkpoint : 이름으로 저장된 체크 포인트 파일 기록
  > model_checkpoint_path: "model.ckpt"
  > alll_model_checkpoint_paths: "model.ckpt"
model.ckpt.data-00000-of-00001 / model.ckpt.index : 학습된 파라미터 저장
model.ckpt.meta : 모델의 그래프 구조 저장(variable, operations, collections 등)

장단기 기억 네트워크(LSTM) 구현

❖ 장단기 기억 네트워크 구현 전반적 구조



장단기 기억 네트워크(LSTM) 구현

❖ 전처리 단계 - Step 1) 파라미터 설정

- 파라미터 설정
  - ✓ 단어의 벡터 차원 수 -> embedding_dim : 100
  - ✓ 레이블 차원 수 -> class_sizes : 2
  - ✓ 문장 내 최대 단어 개수 -> max_sentence_length : 100
  - ✓ LSTM 셀 결과 크기 -> hidden_size : 30
  - ✓ 학습 시 드롭아웃 변수 -> dropout_keep_prob : 0.5
  - ✓ 학습 횟수 -> num_epochs : 20
  - ✓ 패치 사이즈 -> batch_size : 1000
  - ✓ 검증을 위한 조건 -> evaluate_every : 150
  - ✓ 최적화 알고리즘 학습률 -> learn_rate : 0.001
  - ✓ LSTM 셀 레이어 개수 -> num_layers : 2

```
73  def run_lstm(params) :
74      class_sizes = 2
75      max_sentence_length = int(params[1])
76      hidden_size = int(params[2])
77      dropout_keep_prob = float(params[3])
78      num_epochs = int(params[4])
79      batch_size = int(params[5])
80      evaluate_every = int(params[6])
81      learn_rate = float(params[7])
82      num_layers = int(params[8])
```

장단기 기억 네트워크(LSTM) 구현

❖ 장단기 기억 네트워크 모델 - Step 1) 모델 생성을 위한 변수 초기화

- batch_size : dynamic_rnn에서의 초기 상태를 0으로 초기화하는 데 사용

```
5           # 학습 데이터가 들어갈 플레이스 홀더 선언
6           self.input_x = tf.placeholder(tf.float32, shape=[None, sequence_length, embedding_size], name='input_x')
7           self.input_y = tf.placeholder(tf.float32, shape=[None, num_classes], name='input_y')
8           self.dropout_keep_prob = tf.placeholder(tf.float32, name='dropout_keep_prob')
9           self.batch_size = tf.placeholder(tf.int32, [], name="batch_size")
```

❖ 장단기 기억 네트워크 모델 - Step 2) LSTM 레이어

- BasicLSTMCell 함수 보다 옵션(peephole 연결 등)이 추가된 고급 모델
  - ✓ num_units : LSTM 셀의 유닛 개수로 출력값의 크기
  - ✓ forget_bias : 망각 게이트(forget gate)의 편향(bias)으로 1인 경우 학습 시 망각의 규모를 줄임
  - ✓ state_is_tuple : 튜플의 상태를 true인 경우에는 c_state, m_state를 반환, false의 경우에는 c_state, m_state를 합쳐서 하나로 연결
- 드롭아웃을 적용

```
11          # LSTM Layer
12          with tf.name_scope("lstm"):
13              def lstm_cell():
14                  #tf.nn.rnn_cell.(Basic)LSTMCell / tf.nn.rnn_cell.(Basic)RNNCell / tf.nn.rnn_cell.GRUCell
15                  # LSTM Cell 및 DropOut 설정
16                  lstm = tf.nn.rnn_cell.LSTMCell(num_units=hidden_unit, forget_bias=1.0, state_is_tuple=True)
17                  return tf.nn.rnn_cell.DropoutWrapper(cell=lstm, output_keep_prob=self.dropout_keep_prob)
```

장단기 기억 네트워크(LSTM) 구현

❖ 장단기 기억 네트워크 모델 - Step 2) LSTM 레이어

- MultiRNNCell 함수에서 사용자가 설정한 layer 개수에 따라 LSTM 레이어 생성
- 설정된 LSTM 셀은 dynamic_rnn 함수를 통해 모델의 결과와 마지막 상태 값이 반환
- 출력값은 [batch_size, sequence_length, hidden_unit]의 형태
- 최종 결과는 가장 마지막 결괏값인 [batch_size, hidden_unit]을 사용
- transpose를 사용하여 행렬의 순서를 [sequence_length, batch_size, hidden_unit]의 형태로 변경
- gather로 출력의 마지막 결괏값만 사용하여 [batch_size, hidden_unit] 형태의 값을 저장

```
# RNN Cell을 여러 층 쌓기
lstm_cell = tf.nn.rnn_cell.MultiRNNCell([lstm_cell() for _ in range(num_layer)])
# 초기 state 값을 0으로 초기화
self.initial_state = lstm_cell.zero_state(self.batch_size, tf.float32)
# outputs : [batch_size, sequence_length, hidden_unit]
outputs, state = tf.nn.dynamic_rnn(lstm_cell, self.input_x, initial_state=self.initial_state , dtype=tf.float32)
# output : [sequence_length, batch_sie, hidden_unit)
output = tf.transpose(outputs, [1, 0, 2])
# 마지막 출력만 사용
output = tf.gather(output, int(output.get_shape()[0]) - 1)
```

장단기 기억 네트워크(LSTM) 구현

❖ 장단기 기억 네트워크 모델 학습 실행

- LSTM 모델에 사용할 변수 초기화

```
with tf.Graph().as_default():
    #sess_config = tf.ConfigProto(device_count = {'GPU': 0})
    sess_config = tf.ConfigProto()
    sess_config.gpu_options.allow_growth = True
    sess = tf.Session(config=sess_config)
    with sess.as_default():
        lstm = lstm_model(
            hidden_unit = hidden_size,
            sequence_length=x_train.shape[1],
            num_classes=y_train.shape[1],
            num_layer=num_layers,
            embedding_size=embedding_dim)
```

- 학습 결과
  - ✓ 학습 결과
  - ✓ 총 20회의 학습
  - ✓ 비용 값은 0.395에서 0.316로 감소
  - ✓ 정확도는 82.3%에서 86.3%로 증가

```
def train_step(x_batch, y_batch):
    feed_dict = {
        lstm.input_x: x_batch,
        lstm.input_y: y_batch,
        lstm.batch_size: len(x_batch),
        lstm.dropout_keep_prob: dropout_keep_prob
    }
 def dev_step(x_batch, y_batch, epoch):
     feed_dict = {
         lstm.input_x: x_batch,
         lstm.input_y: y_batch,
         lstm.batch_size: len(x_batch),
         lstm.dropout_keep_prob: 1.0
     }
```

모델 활용

학습된 알고리즘 모델에 대한 검증 구현

- ❖ 합성곱 신경망과 장단기 기억 네트워크 알고리즘으로 학습된 모델을 불러옴
- ❖ 사용자가 선택한 알고리즘 및 테스트 문장을 입력받아 긍정/부정의 감정 결과를 도출
- ❖ Step 1) 변수 초기화
  - python 실행 위치
    - ✓ 웹어플리케이션에서 해당 파일 실행
    - ✓ python 파일의 절대 경로 기준으로 파일 위치 설정
  - word2vec 모델 파일 위치
  - 학습된 CNN, LSTM 알고리즘 저장 위치

```
7   # py 파일이 실행되는 폴더 위치
8   base_dir = os.path.dirname(os.path.realpath(__file__)) + '/'
9   source_dir ='./data/'
10  # word2vec 파일 이름
11  w2v_file_name = '3_word2vec_nsmc.w2v'
12  # 알고리즘 학습  모델 저장 경로
13  cnn_model_dir = './cnn_model'
14  lstm_model_dir = './lstm_model'
```

학습된 알고리즘 모델에 대한 검증 구현

❖ Step 2) 사용자가 입력할 문장에 대한 전처리

- 네이버 맞춤법 검사 -> 품사 부착 -> 단어에 대한 벡터 변환

```
16  # 사용자 입력 문장에 대해 단어 벡터 변환
17  def data_setting(w2v_model, embedding_dim, max_word_length, evaluation_text):
18      eval_arrays = np.zeros((1, max_word_length, embedding_dim))
19      # 네이버 맞춤법 검사 적용
20      eval_spell_chcker = pre.naver_spell_cheker(evaluation_text)
21      # 품사 부착 진행
22      eval_pos_tag = pre.konlpy_pos_tag(eval_spell_chcker)
23      #문장 내 단어 벡터 변환
24      eval_arrays[0] = pre.max_word_length_word2vec(w2v_model, embedding_dim, max_word_length, eval_pos_tag)
25
26      return eval_arrays
```

학습된 알고리즘 모델에 대한 검증 구현

❖ Step 3) 학습된 알고리즘에 대한 검증 - 변수 초기화

- CNN, LSTM 모델 학습 시 사용했던 문장 내 최대 단어 길이
  - ✓ 단어 길이를 다르게 설정하면 텐서 형태가 다르다는 에러 발생
- 사용자가 입력한 문장
- word2vec 모델에서의 vector 크기

```
28  def evaluation(params) :
29      # 각 알고리즘 별 최대 단어 개수 지정
30      # 모델 학습 시 사용했던 값 사용
31      cnn_max_sentence_length = 50
32      lstm_max_sentence_length = 100
33      evaluation_text = params[2]
34
35      w2v_model = Word2Vec.load(base_dir + source_dir + w2v_file_name)
36      embedding_dim = w2v_model.vector_size
```

학습된 알고리즘 모델에 대한 검증 구현

❖ Step 3) 학습된 알고리즘에 대한 검증

- 전처리 데이터 저장 및 저장된 모델 파일 위치 저장

```
38      # 알고리즘 별 데이터 셋팅 및 모델의 마지막 저장된 checkpoint 파일 이름 검색
39      if params[1] == 'CNN' :
40          x_eval = data_setting(w2v_model, embedding_dim, cnn_max_sentence_length, evaluation_text)
41          checkpoint_file = tf.train.latest_checkpoint(base_dir + cnn_model_dir)
42
43      elif params[1] == 'LSTM' :
44          x_eval = data_setting(w2v_model, embedding_dim, lstm_max_sentence_length, evaluation_text)
45          checkpoint_file = tf.train.latest_checkpoint(base_dir + lstm_model_dir)
```

- 학습된 모델의 Tensorflow graph 저장 및 모델 로드
  - ✓ graph에서는 변수, 오퍼레이션, 컬렉션이 저장되어 있음

```
with sess.as_default():
    # 저장된 그래프를 재생성하여 모델을 불러옴
    # Tensorflow graph를 저장 하게 된다. 즉 all variables, operations, collections 등을 저장 한다. .meta로 확장자를 가진다.
    saver = tf.train.import_meta_graph("{}.meta".format(checkpoint_file))
    saver.restore(sess, checkpoint_file)
```

학습된 알고리즘 모델에 대한 검증 구현

❖ Step 3) 학습된 알고리즘에 대한 검증

- 그래프 내 오퍼레이션 확인

```
# 그래프 내 operation 리스트 확인
#for op in graph.get_operations():
#    print(op.name)
```

```
input_x
input_y
dropout_keep_prob
ExpandDims/dim
ExpandDims
conv-maxpool-2/truncated_normal/shape
conv-maxpool-2/truncated_normal/mean
conv-maxpool-2/truncated_normal/stddev
… 이하 생략
```

학습된 알고리즘 모델에 대한 검증 구현

❖ Step 3) 학습된 알고리즘에 대한 검증

- 그래프 내에서의 오퍼레이션 불러오기
  - ✓ 모델에 넣을 변수 - input_x, dropout_keep_prob, predictions, result
  - ✓ feed_dict를 모델에 사용될 실제 값 입력

```
# 그래프에서의 Operation 불러오기
input_x = graph.get_operation_by_name("input_x").outputs[0]
dropout_keep_prob = graph.get_operation_by_name("dropout_keep_prob").outputs[0]
predictions = graph.get_operation_by_name("output/predictions").outputs[0]
result = graph.get_operation_by_name("output/result").outputs[0]

if params[1] == 'CNN' :
    feed_dict = {input_x: x_eval, dropout_keep_prob: 1.0}
elif params[1] == 'LSTM' :
    batch_size = graph.get_operation_by_name("batch_size").outputs[0]
    feed_dict = {input_x: x_eval, batch_size: 1, dropout_keep_prob: 1.0}
```

학습된 알고리즘 모델에 대한 검증 구현

❖ Step 3) 학습된 알고리즘에 대한 검증

- 모델을 통한 입력 값 검증, 결과 값 출력
  - ✓ 결과 예측 값이 1인 경우 긍정, 0인 경우 부정
  - ✓ Softmax 를 통해 나온 값으로 확률 값 측정
  - ✓ 웹어플리케이션에 출력될 문장에 대해 print 문 작성

```
eval_pred, eval_result = sess.run([predictions, result], feed_dict)

# 예측된 결과에 대해 긍정/부정으로 나누고 Softmax를 통해 나온 값을 통해 확률 계산
result_pred = '긍정' if(eval_pred == 1) else '부정'
result_score = eval_result[0][1] if(eval_pred == 1) else eval_result[0][0]

print('입력된 [' + evaluation_text + ']는 ')
print('[' + str('{:.2f}'.format(result_score * 100)) + ']%의 확률로 [' + result_pred + ']으로 예측됩니다.')
```

학습된 알고리즘 모델에 대한 검증 구현

❖ Step 3) 학습된 알고리즘에 대한 검증

- 모델을 통한 입력 값 검증, 결과 값 출력
  - ✓ 결과 예측 값이 1인 경우 긍정, 0인 경우 부정
  - ✓ Softmax 를 통해 나온 값으로 확률 값 측정
  - ✓ 웹어플리케이션에 출력될 문장에 대해 print 문 작성

```
eval_pred, eval_result = sess.run([predictions, result], feed_dict)

# 예측된 결과에 대해 긍정/부정으로 나누고 Softmax를 통해 나온 값을 통해 확률 계산
result_pred = '긍정' if(eval_pred == 1) else '부정'
result_score = eval_result[0][1] if(eval_pred == 1) else eval_result[0][0]

print('입력된 [' + evaluation_text + ']는 ')
print('[' + str('{:.2f}'.format(result_score * 100)) + ']%의 확률로 [' + result_pred + ']으로 예측됩니다.')
```

[합성곱 신경망 예측 결과]
입력된 [다시 찾아 보고 싶은 영화입니다.]는
[94.78]%의 확률로 [긍정]으로 예측됩니다.

[장단기 기억 네트워크 예측 결과]
입력된 [다시 찾아 보고 싶은 영화입니다.]는
[96.44]%의 확률로 [긍정]으로 예측됩니다.

TENSORFLOW 실행

튜토리얼의 예제 실행 (Tensorflow 2.x 수행)

❖ 구글 코랩을 사용하여 예제가 실행됨 (버전 2.x 가 기본)

- 1.x에서 사용이 쉽게 2.x로 변경

❖ 생성모델 2가지 수행

- 신경 스타일 전이 https://www.tensorflow.org/tutorials/generative/style_transfer
- Deep Dream https://www.tensorflow.org/tutorials/generative/deepdream
- https://github.com/tensorflow/examples/community/en

❖ Data Science에서 제공하는 tensorflow-notebook
(https://www.dataquest.io/blog/docker-data-science/)
$ docker run -p 8888:8888 -v ~/notebooks:/home/jovyan jupyter/tensorflow-notebook

- 노트북 비교 수행