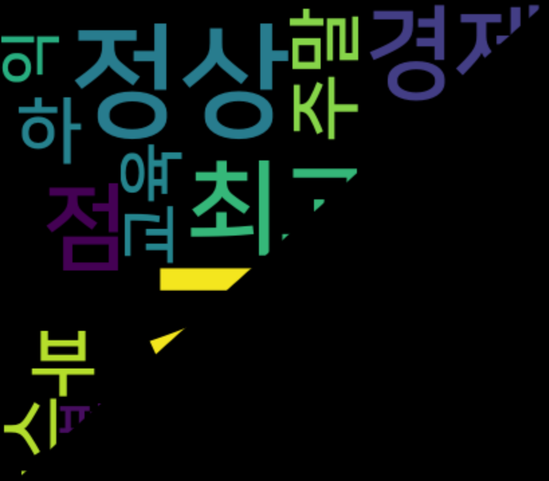
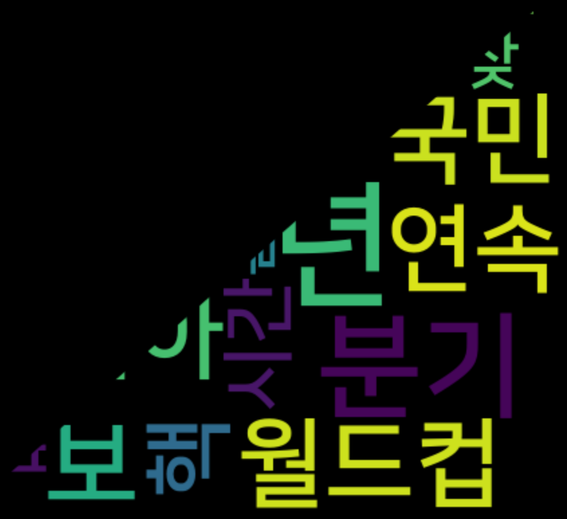
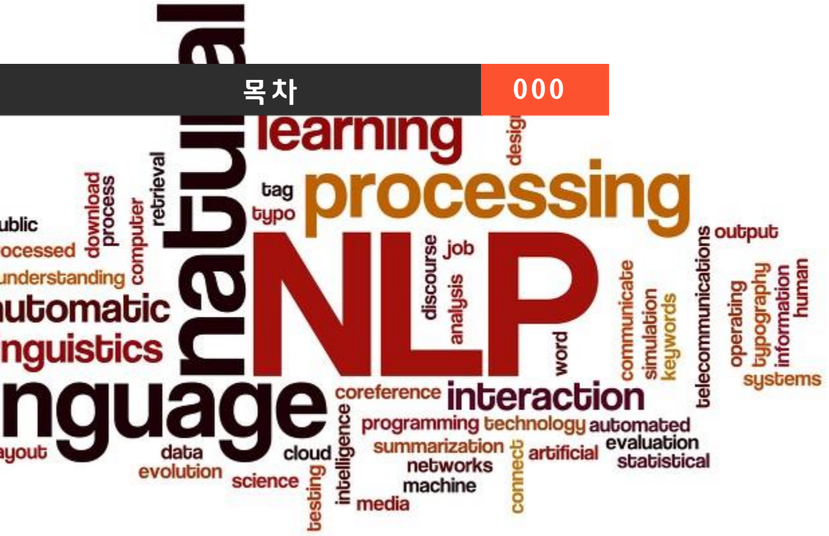
한국어 뉴스 토픽 분류

비타민 7기 자연어

이준석 유진이 유하준 황예원







contents

헤드라인을 이용한 뉴스 토픽 분류

문제 : YNAT 데이터셋의 한국어 뉴스 헤드라인을 이용해, 뉴스의 주제를 분류하는 모델 개발

데이터셋 : YNAT (주제 분류를 위한 연합 뉴스 헤드라인)

	index	title	topic_idx
0	0	인천→핀란드 항공기 결항…휴가철 여행객 분통	4
1	1	실리콘밸리 넘어서겠다…구글 15조원 들여 美전역 거점화	4
2	2	이란 외무 긴장완화 해결책은 미국이 경제전쟁 멈추는 것	4
3	3	NYT 클린턴 측근韓기업 특수관계 조명…공과 사 맞물려종합	4
4	4	시진핑 트럼프에 중미 무역협상 조속 타결 희망	4

	index	title
0	45654	유튜브 내달 2일까지 크리에이터 지원 공간 운영
1	45655	어버이날 맑다가 흐려져…남부지방 옅은 황사
2	45656	내년부터 국가RD 평가 때 논문건수는 반영 않는다
3	45657	김명자 신임 과총 회장 원로와 젊은 과학자 지혜 모을 것
4	45658	회색인간 작가 김동식 양심고백 등 새 소설집 2권 출간

train_data.csv

- index : 헤드라인 인덱스
- title : 뉴스 헤드라인
- topic_idx : 뉴스 주제 인덱스 값 (label)

test_data.csv

- index : 헤드라인 인덱스
- title : 뉴스 헤드라인

KoNLPy : 한국어 정보 처리를 위한 패키지

Mecab : 일본어용 형태소 분석기를 한국어에 사용할 수 있도록 수정

```
from konlpy.tag import Mecab
mecab = Mecab()
```

➡ 한국어 POS 태그 참조해 처리

한국어 품사및 태그

아래의 태그는 '세종 계획'에서 나온 태그를 기준으로 하였습니다.
각 library들마다 약간씩 이름이 다를 수 있습니다. 자세한 각 라이브러리별 품사에 대한 태킹을 여기를 눌러주시기 바랍니다.

먼저 용어부터 정리를 하면 다음과 같습니다.

- **체언**: 명사, 대명사, 수사 -> 세 품사를 묶어 체언이라고 함 (예. 사람, 학교, 초등학교, 나, 너, 이것, 하나, 둘째, etc)
- **용언**: 동사, 형용사 두 품사를 묶어 용언이라고 함 (예. 먹다, 달리다, 예쁘다, 착하다)
- **형태소**: 의미를 가진 최소 단위. '사람', '나', '하나' 같은 말은 더 이상 쪼개지지 않습니다.
- **접사**: 독립적으로 쓰이지 못하고, 다른 말에 붙어 새로운 단어를 만드는 형태소. (예. 풋사과, 햇과일, 지우개, 점쟁이)
- **어근**: 단어에서 실질적 의미(중심이 되는)를 나타내는 부분. (사람, 학교, 나, 풋사과, 점쟁이, 초등 학교, 깨끗하다)

tag	name	description	example
NNG	일반 명사	일반적인 사물의 이름을 가르킨다	하늘, 나무, 사랑, 희망
NNP	고유 명사	특정한 사람이나 사물의 이름을 가르킨다	안창호, 금강산, 신라, 한강
NNB	의존 명사	자립명사라고도 하며 스스로 뜻을 지니고 있어 다른 말의 도움없이 쓰이는 명사	뿐, 바, 따름, 이, 데, 줄, 나름, 나위
NR	수사	사물의 수량이나 순서를 나타냄	하나, 둘, 셋, 넷, 다섯, 첫째, 둘째, 셋째
NP	대명사	인칭 대명사는 사람의 이름을 대신하여 가르키며, 지시 대명사는 사람이외의 사물, 장소를 가르키는 말	어르신, 당신, 귀하, 자네, 너, 이것, 저것, 그것, 무엇
VV	동사	동작이나 과정을 나타냄	가다, 먹다, 자다
VA	형용사	사물의 모습이나 상태를 표현	귀엽다, 예쁘다, 노랗다, 둥글다, 있다, 같다
VX	보조 용언	본영언과 연결되어 그것의 뜻을 보충하는 역활. 보조 동사, 보조 형용사등이 있다.	가지고 싶다, 먹어 보다

딕셔너리 형태로 저장 후 해당 태그 품사 제거

```
POS_DICT = {
    'NNB':'의존명사', 'NP':'대명사', 'NR':'수사',
    'VA':'형용사', 'VCP':'긍정 지정사', 'VCN':'부정 지정사', 'VX':'보조 용언',
    'MM':'관형사','MAG':'일반부사', 'MAJ':'접속부사',
    'IC':'감탄사',
    'JKS':'조사', 'JKC':'조사', 'JKG':'조사', 'JKO':'조사', 'JKB':'조사', 'JKV':'조사', 'JKQ':'조사', 'JC':'조사', 'JX':'조사',
    'EP':'어미', 'EF':'어미', 'EC':'어미', 'ETN':'어미', 'ETM':'어미', 'XPN':'어미', 'XSN':'어미', 'XSV':'어미', 'XSA':'어미',
    'SF':'문장부호', 'SE':'문장부호', 'SS':'문장부호'
}
```

제거할 품사 태그

- NNB, NP, NR (의존명사, 대명사, 수사)
- VCP, VCN (긍정, 부정 지정사)
- MM (관형사)
- MAG (일반 부사), MAJ(접속 부사)
- IC (감탄사)
- 조사 : JKS, JKC, JKG, JKO, JKB, JKV, JKQ, JC, JX
- 어미 : EP, EF, EC, ETN, ETM, XPN, XSN, XSV, XSA
- 문장부호 : SF, SE, SS
- 외국어숫자 : SL, SN

한자와 특수기호, 영어 약자 한국어로 텍스트 대체

```
# 전처리를 위한 dictionary
CHN_KOR_DICT = {
    "外人":"외국인", "日":"일본", "美":"미국", "北":"북한", "英":"영국", "中":"중국", "與":"여당", "青":"청와대", "野":"야당", "伊":"이탈리아", "韓":"한국", "南":"한국", "獨":"독일",
    "佛":"프랑스", "檢":"검찰", "銀":"은행", "亞":"아시아", "人":"사람", "孫":"손혜원", "企":"기업", "前":"이전", "反":"반대", "安":"안철수", "展":"전시회", "故":"사망", "文":"문재인",
    "新":"새로운", "曺":"조국", "朴":"박근혜", "株":"주식", "男":"남자", "研":"연구", "車":"자동차", "軍":"군대", "重":"중공업", 'VNL':'배구 네이션스 리그'
}


SPECIAL_WORD_DICT ={r'↑':'상승', r'↓':'하락', r'러':'러시아', r'ℓ':'리터', r'→':'에서 ', r'…':' ',
                    r"[@X₩₩+=()/~#&₩+á?₩xc3₩xa1₩-₩₩,₩:₩:₩₩₩-₩,₩_₩-₩₩₩']":"", r"₩d+":"",  r'①':'',
                    r'②':'', r'⑤':'', r' ':' 및 ', r'㈜':'', r'B52':''
}

ENG_KOR_DICT = {
    'NYT':'미국 일간지','WP':'미국 일간지', 'KAIST':'카이스트', 'WMO':'세계 기상 기구', 'KBL':'한국 프로 농구', 'UAE':'아랍에미리트',
    'EU':'유럽 연합', 'NBA':'농구 연맹', 'CIA':'중앙정보국', 'ECB':'유럽 중앙 은행', 'AFC':'아시아 축구 연맹',
    'ITU':'국제전기통신연합', 'MVP':'최우수 선수', 'APEC':'아시아 태평양 경제협력체','MB':'', 'APEC':'아시아 태평양 경제협력체',
    'PSG':'파리 축구 클럽', 'IMO':'국제해운기구', 'MLB':'프로 야구 리그', 'MOU':'양해각서', 'FA':'자유계약선수제도',
    'EPL':'잉글랜드 축구 리그', 'KBO':'한국야구위원회', 'IPU':'국제 의회 연맹', 'AG':'아시안게임', 'PS':'포스트시즌',
    'PO':'플레이오프', 'OUT':'방출', 'IN':'영입', 'TPP':' 환태평양 경제 동반자협정', 'EAS':'동아시아 정상회의', 'G20':'정상 회의',
    'DC':'', 'ITF':'국제 태권도연맹', 'IS':'이슬람', '₩농구':'한국여자농구', 'C맬리스':'크리스탈맬리스', 'SLBM':'잠수함 발사탄도 미사일',
    'LA타임스':'미국 일간지', 'V리그':'배구 리그', 'KOVO':'한국배구연맹', 'SUN':'', 'WSJ':'미국 일간지',
    'ERA':'야구 방어율', 'IoT':'사물인터넷', 'UFG20':'한미 합동 군사', 'F35':'전투기', 'TK':'대구와 경북', 'ACL':'아시아축구연맹 리그',
    'IT':'정보 기술', 'AI':'인공지능', 'IF':'대책 본부', 'ML':'메이저리그', 'FC':'축구 클럽', 'MS':'마이크로소프트', 'SNS':'소셜 네트워크 서비스',
    'VR':'가상현실', 'ELB':'주가연계파생결합사채', 'CES':'국제 전자제품 박람회', 'NPL':'부실채권', 'IPO':'기업공개', 'ERA':'야구 방어율',
    'MWC':'모바일 산업 박람회', 'NSC':'국가안전보장회의'
}


ETC_WORD_DICT = {
    "종합":"", "그래픽":"", '1보':"", '닷컴':'', 'QS':'', 'NL':'', 'SI':''
}
```

전처리 함수 정의

```python
# 전처리 관련 함수 정의
def clean_text_kor(inputString):
  text_kor = re.sub('[^가-힣]', ' ', inputString)
  text_kor = ' '.join(text_kor.split())
  return text_kor

def get_dict_value(token, dictionary):
    """
        token이 dicionary의 key값으로 존재한다면 해당 value를 반환
    """
    if token in dictionary:
        return dictionary[token]
    else:
        return token

def clean_with_dict(tokenized_title, dictionary):
    """
        title에 포함된 특정 token들이 ditionary의 key값에 존재한다면 변환
    """
    token_list = tokenized_title.split(" ")
    cleaned_list = []
    for token in token_list:
        cleaned_list.append(get_dict_value(token, dictionary))
    return " ".join(cleaned_list)
```

```python
def clean_etc_reg_ex(title):
    """
        정규식을 통해 기타 공백과 기호, 숫자등을 제거
    """
    title = re.sub(r'[@%₩₩*=()/~#&₩+á?₩xc3₩xa1₩-₩|₩.₩:₩;₩!₩-₩,₩_₩~₩$₩'₩"]', '', title) #remove punctuation
    title = re.sub(r'[~%①②⑤⑪…→·]', '', title)
    title = re.sub(r'₩d+', '', title) #remove number
    title = re.sub(r'₩s+', ' ', title) #remove extra space
    title = re.sub(r'<[^>]+>','',title) #remove Html tags
    title = re.sub(r'₩s+', ' ', title) #remove spaces
    title = re.sub(r"^₩s+", '', title) #remove space from start
    title = re.sub(r'₩s+$', '', title) #remove space from the end
    title = re.sub("[——顧]",'', title)
    return title

def clean_with_pos(title):
    """
        품사 종류를 바탕으로 단어를 제거
    """
    tagged_tuples = tokenizer.pos(title)
    tmp = []
    for tag in tagged_tuples:
        if tag[1] not in POS_DICT:
            tmp.append(tag[0])
    return " ".join(tmp)
```

전처리 전/후 wordcloud 비교

	train	test
전		
후		

딥러닝 : 단순 전처리 + Simple Dense Layer 모델

```
# 데이터 전처리

def clean_text(sent):
  sent_clean = re.sub('[^가-힣ㄱ-ㅎㅏ-ㅣ\\s]',' ',sent)
  return sent_clean

train['cleaned_title'] = train['title'].apply(lambda x: clean_text(x))
test['cleaned_title'] = test['title'].apply(lambda x: clean_text(x))

# 값을 리스트로 변환
train_text = train['cleaned_title'].tolist()
test_text = test['cleaned_title'].tolist()

# array로 타입 변경
train_label = np.asarray(train.topic_idx)

# tf-idf
tfidf = TfidfVectorizer(analyzer='word',sublinear_tf=True,ngram_range=(1,2),max_features=150000, binary= False )
train_tf_text = tfidf.fit_transform(train_text).astype('float32') # 결과 찍어보면, '<class 'numpy.float32'>' 이
test_tf_text = tfidf.transform(test_text).astype('float32') # test set은 transform만 시킴 # fit을 시키면, test d
# train_tf_text.shape
# test_tf_text.shape
```

```
# model_ 기본 dnn_model

def dnn_model():
  model = Sequential()
  model.add(Dense(128, input_dim = 150000, activation='relu'))
  model.add(Dropout(0.8))
  model.add(Dense(7,activation='softmax'))
  return model

model = dnn_model()
model.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = tf.optimizers.Adam(learning_rate=0.001),
              metrics = ['accuracy'])
```

```
# 모델의 history 살펴보기
history = model.fit(x = train_tf_text[:40000],y = train_label[:40000],
                    validation_data = (train_tf_text[40000:],train_label[40000:]),
                    epochs = 4)
```

```
Epoch 1/4
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/indexed_slices.py:450: UserWarning: Converting sparse IndexedSlices(Ind
  "shape. This may consume a large amount of memory." % value)
1250/1250 [==============================] - 233s 186ms/step - loss: 1.3988 - accuracy: 0.5444 - val_loss: 1.1084 - val_accuracy: 0.7274
Epoch 2/4
1250/1250 [==============================] - 232s 186ms/step - loss: 0.6782 - accuracy: 0.8031 - val_loss: 0.7667 - val_accuracy: 0.7713
Epoch 3/4
1250/1250 [==============================] - 232s 185ms/step - loss: 0.4392 - accuracy: 0.8727 - val_loss: 0.6748 - val_accuracy: 0.7837
Epoch 4/4
1250/1250 [==============================] - 232s 186ms/step - loss: 0.3171 - accuracy: 0.9098 - val_loss: 0.6494 - val_accuracy: 0.7849
```

사용된 하이퍼 파라미터

- dense layer :128
- drop out : 0.2
- activation function : softmax
- loss fuction : sparse_categorical_crossentropy
- optimizer : adam

딥러닝 : 정교한 전처리 + Simple Dense Layer 모델

```
# tf-idf
tfidf = TfidfVectorizer(analyzer='word',sublinear_tf=True,ngram_range=(1,2),max_features=150000,binary= False )
train_tf_text = tfidf.fit_transform(train_text).astype('float32') # 결과 찍어보면, '<class 'numpy.float32'>' 이
test_tf_text = tfidf.transform(test_text).astype('float32') # test set은 transform만 시킴 # fit을 시키면, test d
# train_tf_text.shape
# test_tf_text.shape
```

```
# model_ 기본 dnn_model로 돌려봄

def dnn_model():
  model = Sequential()
  model.add(Dense(128, input_dim = 150000, activation='relu'))
  model.add(Dropout(0.8))
  model.add(Dense(7,activation='softmax'))
  return model

model = dnn_model()
model.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = tf.optimizers.Adam(learning_rate=0.001),
              metrics = ['accuracy'])
```

```
# 모델의 history 살펴보기
history = model.fit(x = train_tf_text[:40000],y = train_label[:40000],
                    validation_data = (train_tf_text[40000:],train_label[40000:]),
                    epochs = 4)                                          # 레이어를 깊게 쌓을경우,epoch을 더 많
```

```
Epoch 1/4
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/indexed_slices.py:450: UserWarning: Converting sparse IndexedSlices(In
  "shape. This may consume a large amount of memory." % value)
1250/1250 [==============================] - 246s 196ms/step - loss: 1.1925 - accuracy: 0.6320 - val_loss: 0.8270 - val_accuracy: 0.8012
Epoch 2/4
1250/1250 [==============================] - 247s 197ms/step - loss: 0.5433 - accuracy: 0.8435 - val_loss: 0.5705 - val_accuracy: 0.8261
Epoch 3/4
1250/1250 [==============================] - 247s 198ms/step - loss: 0.3740 - accuracy: 0.8906 - val_loss: 0.5229 - val_accuracy: 0.8300
Epoch 4/4
1250/1250 [==============================] - 246s 197ms/step - loss: 0.2833 - accuracy: 0.9186 - val_loss: 0.5014 - val_accuracy: 0.8327
```

사용된 하이퍼 파라미터 : 앞 모델과 동일

Prepro_Simple_DNN.csv	0.821686747	
edit	0.7901883487	
Simple_DNN.csv	0.7877327492	
edit	0.7568988173	

단순 전처리

- layer의 개수를 늘려도 성능의 변화가 크게 없음
- overfitting을 방지하기 위해 dropout을 사용
- categorical entropy, sparser categorical entropy의 input type이 다름 → one-hot vector로 사용
- val accuracy : 0.78
- test accuracy
  - Public : 0.75
  - Private : 0.78

정교화 전처리

- validation, test accuracy에서 모두 성능 향상을 보임
- val accuracy : 0.83
- test accuracy :
  - public : 0.79
  - private : 0.82

전처리만 잘해도 좋은 성능 향상을 기대할 수 있음

모델링

01. RNN

02. Bi-LSTM
    a. Bi-LSTM
    b. Bi-LSTM (+Fasttext)
    c. Bi-LSTM & CNN ensemble

03. BERT
    a. KoBERT (1)
    b. KoBERT (2)
    c. DistilBert

RNN

Konlpy의 형태소 토크나이징을 통해 (동사, 명사)로 전처리된 피처 활용

```
# 먼저 train 데이터와 test 데이터 인덱스 없이 배열로 만들기
X_train = np.array([x for x in train_merge['title_detok_morph']])
X_test = np.array([x for x in test['title_detok_morph']])
Y_train = np.array([x for x in train_merge['topic_idx']])

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
```

```
(45654,)
(9131,)
(45654,)
```

```
print(X_train)
print(X_test)
print(Y_train)
```

```
['인천 핀란드 항공기 결항 휴가철 여행객 분통' '실리콘밸리 넘어서 구글 조원 들여 미국 전역 거점'
 '이란 외무 긴장 완화 해결책 미국 경제 전쟁 멈추' ... '게시판 키움증권 키움 영웅전 실전 투자 대회'
 '답변 배기동 국립 중앙 박물 관장' '한국 인터넷 기자상 시상식 내달 일 개 특별상 김성후']
['유튜브 내달 일 크리에이터 지원 공간 운영' '어버이날 맑 흐려져 남부 지방 옅 황사' '내년 국가 평가 때 논문 건수 반영 않'
 ... '년 전 부마 항쟁 부산 시위 사진 점 최초 공개' '게시판 아리랑 아프리카 개발은행 총회 개회식 중계'
 '유영민 과기 장관 강소 특구 지역 혁신 중심 지원 책 강구']
[4 4 4 ... 1 2 2]
```

RNN

등장 횟수가 1회인 단어들은 자연어 처리에서 배제 → 어느정도 비중을 차지하는지 확인

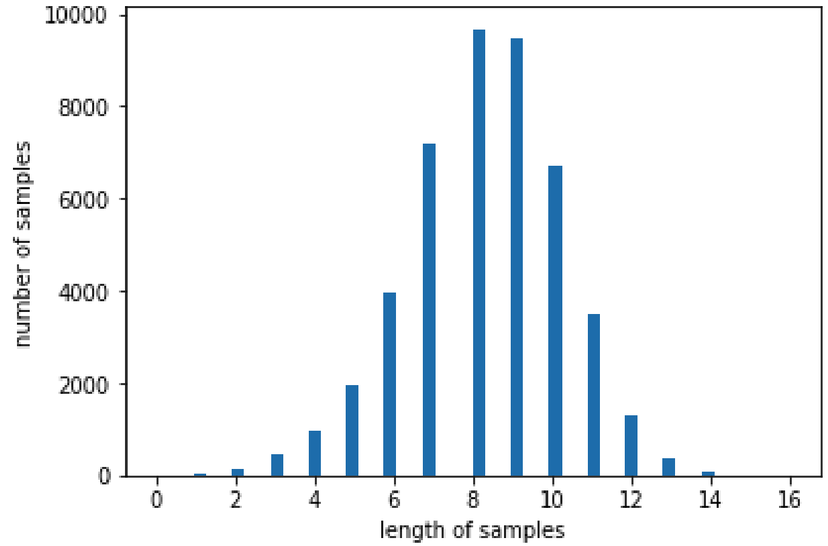
```
[189] # Tokenizer
      from keras.preprocessing.text import Tokenizer
      vocab_size = 2000

      tokenizer = Tokenizer(num_words = vocab_size)
        # Tokenizer 는 데이터에 출현하는 모든 단어의 개수를 세고 빈도 수로 정렬해서
        # num_words 에 지정된 만큼만 숫자로 반환하고, 나머지는 0 으로 반환합니다
      tokenizer.fit_on_texts(X_train) # Tokenizer 에 데이터 실제로 입력
      sequences_train = tokenizer.texts_to_sequences(X_train)     # 문장 내 모든 단어를 시퀀스 번호로 변환
      sequences_test = tokenizer.texts_to_sequences(X_test)       # 문장 내 모든 단어를 시퀀스 번호로 변환

      print(len(sequences_train), len(sequences_test))
```

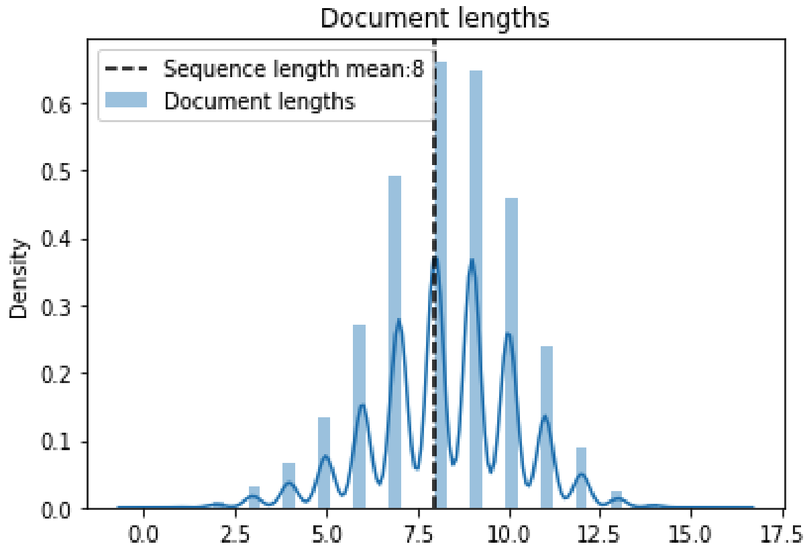
```
단어 집합(vocabulary)의 크기 : 28002
등장 빈도가 1번 이하인 희귀 단어의 수: 10364
단어 집합에서 희귀 단어의 비율: 37.0116420255696
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 2.7398419118618977
```

RNN

전체 데이터에서 가장 길이가 긴 리뷰와 전체 데이터 길이 분포 확인

```
리뷰의 최대 길이 : 16
리뷰의 평균 길이 : 8.28558286239979
```





가장 긴 문장은 16 개의 단어를, 가장 짧은 문장은 0 개의 단어를 가지고 있습니다.

RNN

훈련용 리뷰를 길이 16으로 패딩

```
# 독립변수 데이터 전처리
  ## 문장의 길이기 제각각이기 때문에 벡터 크기 다 다름
  ## 그러므로 최대 시퀀스 길이 크기(211) 만큼 넉넉하게 늘리고
  ## 패딩(padding) 작업을 통해 나머지 빈 공간을 0으로 채움

max_length = 16     # 위에서 그래프 확인 후 정함
padding_type='post'

train_x = pad_sequences(sequences_train, padding='post', maxlen=max_length)
test_x = pad_sequences(sequences_test, padding=padding_type, maxlen=max_length)

print(train_x.shape, test_x.shape)
```

```
(45654, 16) (9131, 16)
```

RNN

Simple RNN Layer 사용한 모델

(3개의 RNN Layer 사용)

```
#파라미터 설정
vocab_size = 17640 # 위에서 구한 사이즈
embedding_dim = 100
max_length = 16    # 위에서 그래프 확인 후 정함
padding_type='post'
```

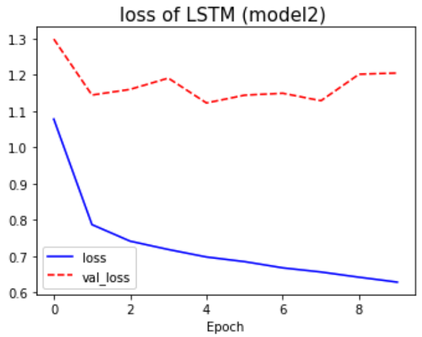
```
# Simple RNN 레이어를 사용한 모델 (model1) 정의
model1 = Sequential([Embedding(vocab_size, embedding_dim, input_length =max_length),
        tf.keras.layers.SimpleRNN(units = 64, return_sequences = True),
        tf.keras.layers.SimpleRNN(units = 64, return_sequences = True),
        tf.keras.layers.SimpleRNN(units = 64),
        Dense(7, activation='softmax')    # 결과값이 0~4 이므로 Dense(5)
    ])

model1.compile(loss= 'categorical_crossentropy',
               #여러개 정답 중 하나 맞추는 문제이므로 손실 함수는 categorical_crossentropy
              optimizer= 'adam',
              metrics = ['accuracy'])
model1.summary()
```

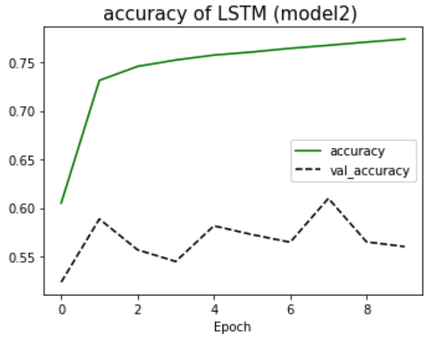
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 16, 100)           1764000

simple_rnn_3 (SimpleRNN)     (None, 16, 64)            10560

simple_rnn_4 (SimpleRNN)     (None, 16, 64)            8256

simple_rnn_5 (SimpleRNN)     (None, 64)                8256

dense_4 (Dense)              (None, 7)                 455

=================================================================
Total params: 1,791,527
Trainable params: 1,791,527
Non-trainable params: 0
_________________________________________________________________
```

RNN

epochs=10, batch_size=100, validation_split= 0.2

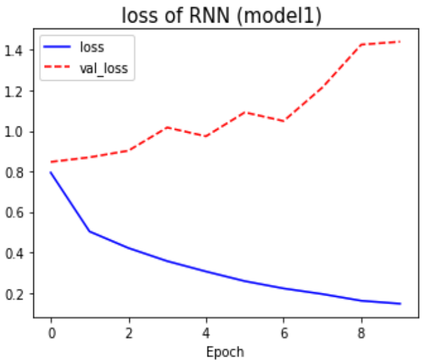
Konlpy 전처리 되기 전 LSTM

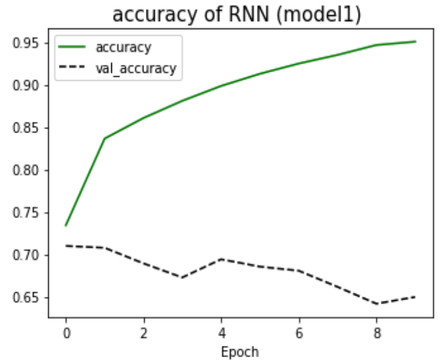





```
Epoch 1/10
366/366 [==============================] - 10s 13ms/step - loss: 1.0782 - accuracy: 0.6051 - val_loss:
1.2989 - val_accuracy: 0.5237
Epoch 2/10
366/366 [==============================] - 4s 10ms/step - loss: 0.7873 - accuracy: 0.7313 - val_loss:
1.1444 - val_accuracy: 0.5887
Epoch 3/10
366/366 [==============================] - 4s 10ms/step - loss: 0.7417 - accuracy: 0.7457 - val_loss:
1.1602 - val_accuracy: 0.5570
Epoch 4/10
366/366 [==============================] - 4s 10ms/step - loss: 0.7187 - accuracy: 0.7523 - val_loss:
1.1912 - val_accuracy: 0.5450
Epoch 5/10
366/366 [==============================] - 4s 10ms/step - loss: 0.6980 - accuracy: 0.7574 - val_loss:
1.1227 - val_accuracy: 0.5816
Epoch 6/10
366/366 [==============================] - 4s 10ms/step - loss: 0.6852 - accuracy: 0.7605 - val_loss:
1.1441 - val_accuracy: 0.5727
Epoch 7/10
366/366 [==============================] - 4s 10ms/step - loss: 0.6681 - accuracy: 0.7643 - val_loss:
1.1494 - val_accuracy: 0.5648
Epoch 8/10
366/366 [==============================] - 4s 10ms/step - loss: 0.6566 - accuracy: 0.7674 - val_loss:
1.1288 - val_accuracy: 0.6099
Epoch 9/10
366/366 [==============================] - 4s 10ms/step - loss: 0.6422 - accuracy: 0.7707 - val_loss:
1.2020 - val_accuracy: 0.5651
Epoch 10/10
366/366 [==============================] - 4s 10ms/step - loss: 0.6287 - accuracy: 0.7739 - val_loss:
1.2051 - val_accuracy: 0.5603
```

Konlpy 전처리 된 후 RNN






```
Epoch 1/10
366/366 [==============================] - 22s 46ms/step - loss: 0.7939 - accuracy: 0.7341 - val_loss: 0.8469 - val_accuracy: 0.7098
Epoch 2/10
366/366 [==============================] - 16s 45ms/step - loss: 0.5029 - accuracy: 0.8363 - val_loss: 0.8696 - val_accuracy: 0.7076
Epoch 3/10
366/366 [==============================] - 27s 74ms/step - loss: 0.4213 - accuracy: 0.8606 - val_loss: 0.9022 - val_accuracy: 0.6892
Epoch 4/10
366/366 [==============================] - 17s 47ms/step - loss: 0.3566 - accuracy: 0.8808 - val_loss: 1.0168 - val_accuracy: 0.6725
Epoch 5/10
366/366 [==============================] - 17s 46ms/step - loss: 0.3053 - accuracy: 0.8984 - val_loss: 0.9735 - val_accuracy: 0.6940
Epoch 6/10
366/366 [==============================] - 18s 49ms/step - loss: 0.2576 - accuracy: 0.9127 - val_loss: 1.0917 - val_accuracy: 0.6854
Epoch 7/10
366/366 [==============================] - 17s 45ms/step - loss: 0.2215 - accuracy: 0.9248 - val_loss: 1.0488 - val_accuracy: 0.6805
Epoch 8/10
366/366 [==============================] - 17s 46ms/step - loss: 0.1937 - accuracy: 0.9349 - val_loss: 1.2147 - val_accuracy: 0.6615
Epoch 9/10
366/366 [==============================] - 17s 47ms/step - loss: 0.1607 - accuracy: 0.9466 - val_loss: 1.4264 - val_accuracy: 0.6417
Epoch 10/10
366/366 [==============================] - 17s 47ms/step - loss: 0.1461 - accuracy: 0.9506 - val_loss: 1.4409 - val_accuracy: 0.6495
```

RNN

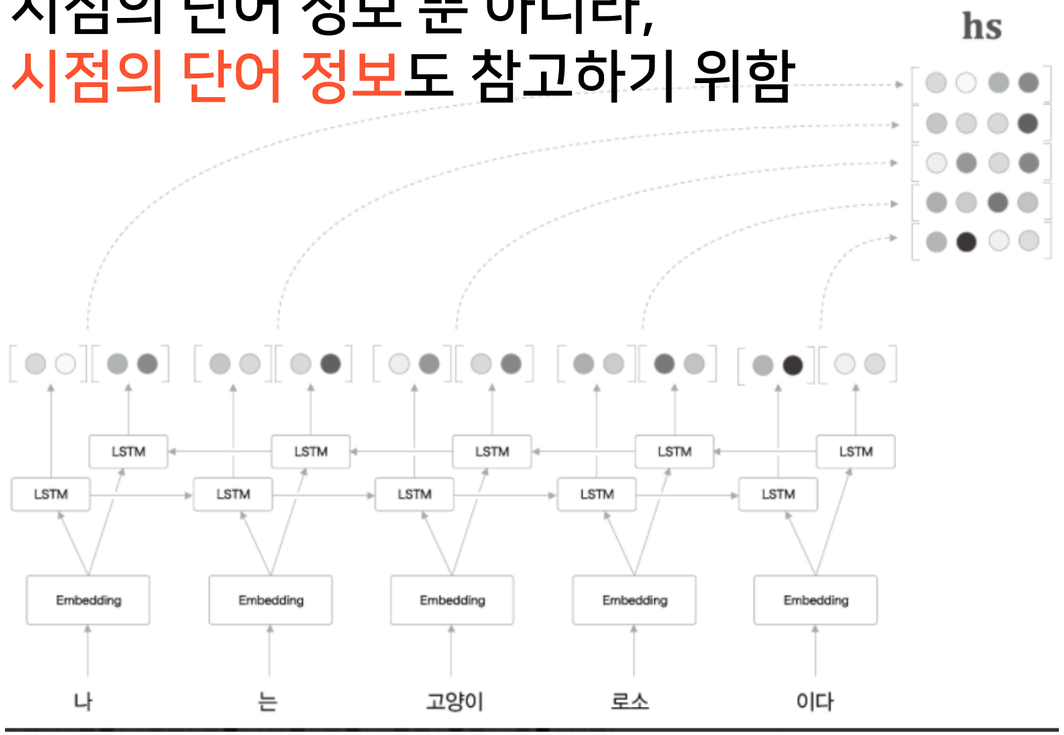
DACON 제출 결과

20220201_RNN_2.csv edit	2022-02-03 23:58:41	0.7886089814 0.7520805957

Bi-LSTM

양방향 LSTM을 사용 (Bi-LSTM, Ko et al., 2018)

이전 시점의 단어 정보 뿐 아니라,
다음 시점의 단어 정보도 참고하기 위함



Bi-LSTM

Bi-LSTM Layer 사용한 모델

(3개의 Bi-LSTM Layer 사용)

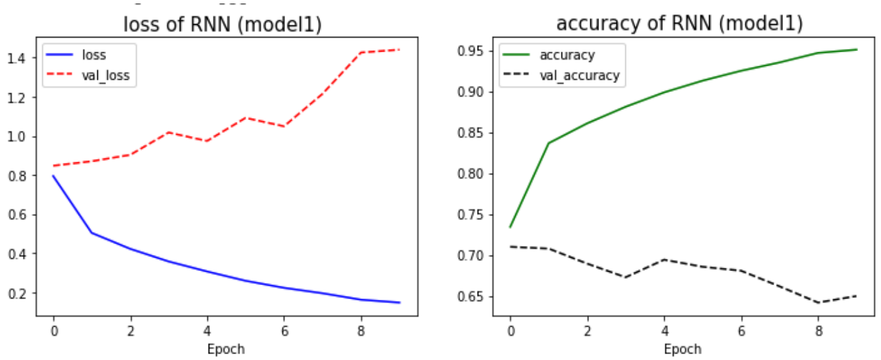
```
# 양방향 LSTM 레이어를 사용한 모델 (model3) 정의
model3 = Sequential([Embedding(vocab_size, embedding_dim, input_length =max_length),
        tf.keras.layers.Bidirectional(LSTM(units = 64, return_sequences = True)),
        tf.keras.layers.Bidirectional(LSTM(units = 64, return_sequences = True)),
        tf.keras.layers.Bidirectional(LSTM(units = 64)),
        Dense(7, activation='softmax')    # 결과값이 0~4 이므로 Dense(5)
    ])

model3.compile(loss= 'categorical_crossentropy', # 여러개 정답 중 하나 맞추는 문제이므로 손실 함
              optimizer= 'adam',
              metrics = ['accuracy'])
model3.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 16, 100)           1764000

 bidirectional_9 (Bidirectio  (None, 16, 128)          84480
 nal)

 bidirectional_10 (Bidirecti  (None, 16, 128)          98816
 onal)

 bidirectional_11 (Bidirecti  (None, 128)              98816
 onal)

 dense_5 (Dense)             (None, 7)                 903

=================================================================
Total params: 2,047,015
Trainable params: 2,047,015
Non-trainable params: 0
_________________________________________________________________
```

Bi-LSTM

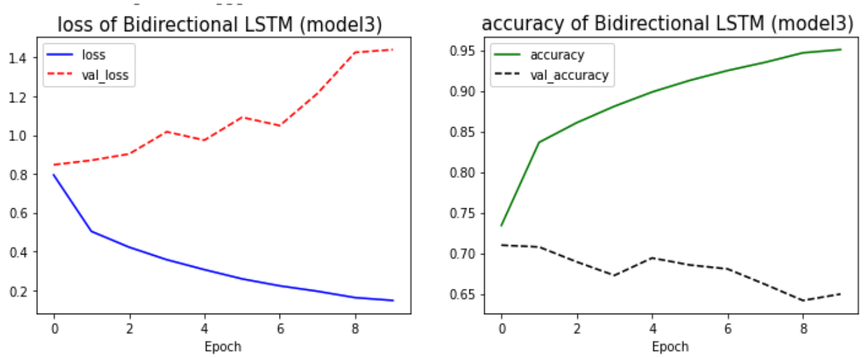
epochs=10, batch_size=100, validation_split= 0.2

Konlpy 전처리 된 후 RNN




```
Epoch 1/10
366/366 [==============================] - 22s 46ms/step - loss: 0.7939 - accuracy: 0.7341 - val_loss: 0.8469 - val_accuracy: 0.7098
Epoch 2/10
366/366 [==============================] - 16s 45ms/step - loss: 0.5029 - accuracy: 0.8363 - val_loss: 0.8696 - val_accuracy: 0.7076
Epoch 3/10
366/366 [==============================] - 27s 74ms/step - loss: 0.4213 - accuracy: 0.8606 - val_loss: 0.9022 - val_accuracy: 0.6892
Epoch 4/10
366/366 [==============================] - 17s 47ms/step - loss: 0.3566 - accuracy: 0.8808 - val_loss: 1.0168 - val_accuracy: 0.6725
Epoch 5/10
366/366 [==============================] - 17s 46ms/step - loss: 0.3053 - accuracy: 0.8984 - val_loss: 0.9735 - val_accuracy: 0.6940
Epoch 6/10
366/366 [==============================] - 18s 49ms/step - loss: 0.2576 - accuracy: 0.9127 - val_loss: 1.0917 - val_accuracy: 0.6854
Epoch 7/10
366/366 [==============================] - 17s 45ms/step - loss: 0.2215 - accuracy: 0.9248 - val_loss: 1.0488 - val_accuracy: 0.6805
Epoch 8/10
366/366 [==============================] - 17s 46ms/step - loss: 0.1937 - accuracy: 0.9349 - val_loss: 1.2147 - val_accuracy: 0.6615
Epoch 9/10
366/366 [==============================] - 17s 47ms/step - loss: 0.1607 - accuracy: 0.9466 - val_loss: 1.4264 - val_accuracy: 0.6417
Epoch 10/10
366/366 [==============================] - 17s 47ms/step - loss: 0.1461 - accuracy: 0.9506 - val_loss: 1.4409 - val_accuracy: 0.6495
```

Bi-LSTM




```
Epoch 19/20
366/366 [==============================] - 69s 187ms/step - loss: 0.1327 - accuracy: 0.9538 - val_loss: 1.6017 - val_accuracy: 0.6932
Epoch 20/20
366/366 [==============================] - 68s 185ms/step - loss: 0.1256 - accuracy: 0.9559 - val_loss: 1.6632 - val_accuracy: 0.6898
```

Bi-LSTM

- Bi-LSTM 모델을 활용
- 계층 교차검증(StratifiedKFold) 적용하고 모델 일반화
- val_loss 기준으로 조기종료 옵션을 추가

```python
# 계층 교차 검증
n_fold = 5
seed = 42

cv = StratifiedKFold(n_splits = n_fold, shuffle=True, random_state=seed)

# 테스트데이터의 예측값 담을 곳 생성
test_y = np.zeros((test_x.shape[0], 7))

# 조기 종료 옵션 추가
es = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=3,
                   verbose=1, mode='min', baseline=None, restore_best_weights=True)

for i, (i_trn, i_val) in enumerate(cv.split(train_x, Y_train), 1):
    print(f'training model for CV #{i}')

    model3.fit(train_x[i_trn],
            to_categorical(Y_train[i_trn]),
            validation_data=(train_x[i_val], to_categorical(Y_train[i_val])),
            epochs=10,
            batch_size=512,
            callbacks=[es])     # 조기 종료 옵션

    test_y += model3.predict(test_x) / n_fold    # 나온 예측값들을 교차 검증 횟수로 나눈다
```

```
training model for CV #5
Epoch 1/10
72/72 [==============================] - 42s 581ms/step - loss: 0.4265 - accuracy: 0.8571 - val_loss: 0.4275 - val_accuracy: 0.8567
Epoch 2/10
72/72 [==============================] - 42s 579ms/step - loss: 0.4040 - accuracy: 0.8656 - val_loss: 0.4575 - val_accuracy: 0.8453
Epoch 3/10
72/72 [==============================] - 42s 584ms/step - loss: 0.3903 - accuracy: 0.8683 - val_loss: 0.4817 - val_accuracy: 0.8394
Epoch 4/10
72/72 [==============================] - ETA: 0s - loss: 0.3775 - accuracy: 0.8737Restoring model weights from the end of the best epoch: 1.
72/72 [==============================] - 43s 596ms/step - loss: 0.3775 - accuracy: 0.8737 - val_loss: 0.5045 - val_accuracy: 0.8321
Epoch 00004: early stopping
```

Bi-LSTM & FastText

```
!pip3 install fasttext
import fasttext
```

```
# fast text 의 인풋은 파일 경로가 되어야 한다
# input must be a filepath. The input text does not need to be tokenized
# as per the tokenize function, but it must be preprocessed and encoded as UTF-8.

model = fasttext.train_supervised('./data_train.txt', dim = 100, ws = 3, minCount = 100)
```

```
#정수인덱싱과정
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data_train)
vocab_size=len(tokenizer.word_index)+1
data_train = tokenizer.texts_to_sequences(data_train)
data_test = tokenizer.texts_to_sequences(data_test)

#모든 문장에서 가장 긴 단어 벡터 길이 구하기
max_len=max(len(l) for l in data_train)

#max_len에 맞춰서 패딩하기
X_train = pad_sequences(data_train, maxlen = max_len)
X_test = pad_sequences(data_test, maxlen = max_len)
```

- python의 FastText 라이브러리를 사용해 형태소 전처리 된 칼럼을 임베딩 진행

- FastText는 하나의 단어 안에도 여러 단어들이 존재하는 것으로 간주, 서브워드(subword)를 고려하여 학습

```
array([[    0,     0,     0, ...,  5302,  2558,  8262],
       [    0,     0,     0, ...,     5,  1575,  3026],
       [    0,     0,     0, ...,    68,   387,  3922],
       ...,
       [    0,     0,     0, ...,  2111,    36,    82],
       [    0,     0,     0, ...,   670, 12284,  4067],
       [    0,     0,     0, ...,    29, 10106, 28002]], dtype=int32)


array([[    0,     0,     0, ...,    44,  1441,   330],
       [    0,     0,     0, ...,   373, 11313,  3090],
       [    0,     0,     0, ...,  4354,  2216,   106],
       ...,
       [    0,     0,     0, ...,   108,   551,    61],
       [    0,     0,     0, ...,  1439,  5196,  1077],
       [    0,     0,     0, ...,    44,   417,  4372]], dtype=int32)
```

Bi-LSTM & FastText

Bi-LSTM Layer 사용한 모델 (3개의 Layer)

```
max_len=16
# 아래 그래프에서 max length 16으로 설정
vocab_size=28003
embedding_dim = 100
model_5=Sequential([
Embedding(vocab_size, embedding_dim, weights=[embedding_matrix],input_length=max_len,trainable = True),
tf.keras.layers.Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2,return_sequences=True)),
tf.keras.layers.Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2,return_sequences=True)),
tf.keras.layers.Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2)),
Dense(7, activation='softmax')])
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 16, 100)           2800300

 bidirectional_3 (Bidirectio  (None, 16, 128)          84480
 nal)

 bidirectional_4 (Bidirectio  (None, 16, 128)          98816
 nal)

 bidirectional_5 (Bidirectio  (None, 128)              98816
 nal)

 dense_1 (Dense)             (None, 7)                 903

=================================================================
Total params: 3,083,315
Trainable params: 3,083,315
Non-trainable params: 0
_________________________________________________________________
```

사용된 하이퍼 파라미터

- dropout : 0.2
- Recurrent_dropout : 0.2
- activation function : softmax
- loss fuction : categorical_crossentropy
- optimizer : adam

Bi-LSTM & FastText

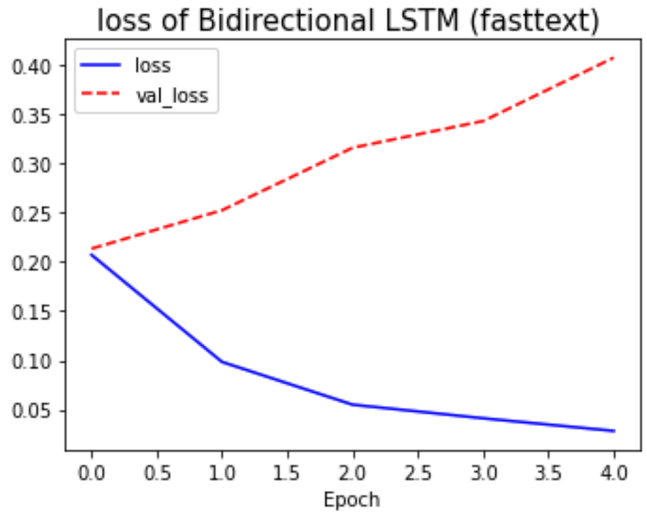
Early stopping 및 model checkpoint 추가

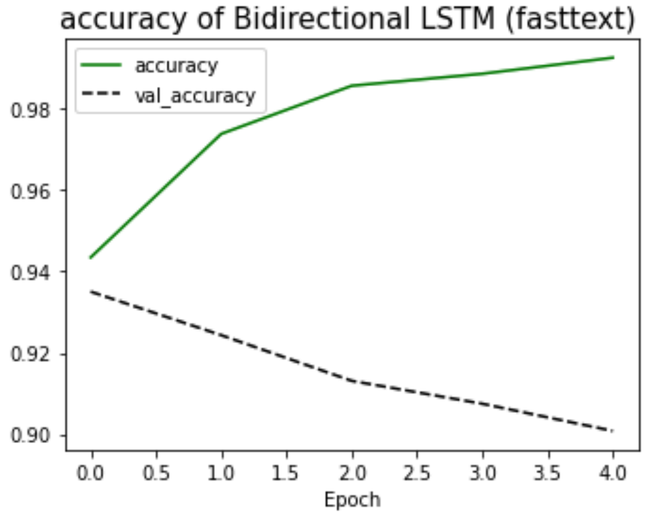
```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=1000)

from tensorflow.keras.utils import to_categorical
model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_5.fit(X_train, y_train, epochs=15, callbacks=[es, mc],
                      validation_data=(X_valid,y_valid), batch_size =100, validation_split = 0.2)
```

너무 많은 epoch → overfitting

너무 적은 epoch → underfitting

Bi-LSTM & FastText






```
Epoch 4/15
150/150 [==============================] - ETA: 0s - loss: 0.0412 - accuracy: 0.9886WARNING:tensorflow:Can save best model only with va
150/150 [==============================] - 49s 326ms/step - loss: 0.0412 - accuracy: 0.9886 - val_loss: 0.3431 - val_accuracy: 0.9075
Epoch 5/15
150/150 [==============================] - ETA: 0s - loss: 0.0284 - accuracy: 0.9925WARNING:tensorflow:Can save best model only with va
150/150 [==============================] - 48s 322ms/step - loss: 0.0284 - accuracy: 0.9925 - val_loss: 0.4074 - val_accuracy: 0.9008
Epoch 00005: early stopping
```

단어 그대로 정수 인코딩하여 임베딩한 것보다
FastText의 가중치 임베딩 처리가
더 나은 성능을 보임

Bi-LSTM & FastText

DACON 제출 결과

20220201_BILSTM.csv edit	2022-02-03 09:39:48	0.7875136911 0.7678493211
20220201_Fasttext와 3개_BILSTM계층.csv edit	2022-02-02 21:00:04	0.7805038335 0.7380639509

Bi-LSTM & CNN

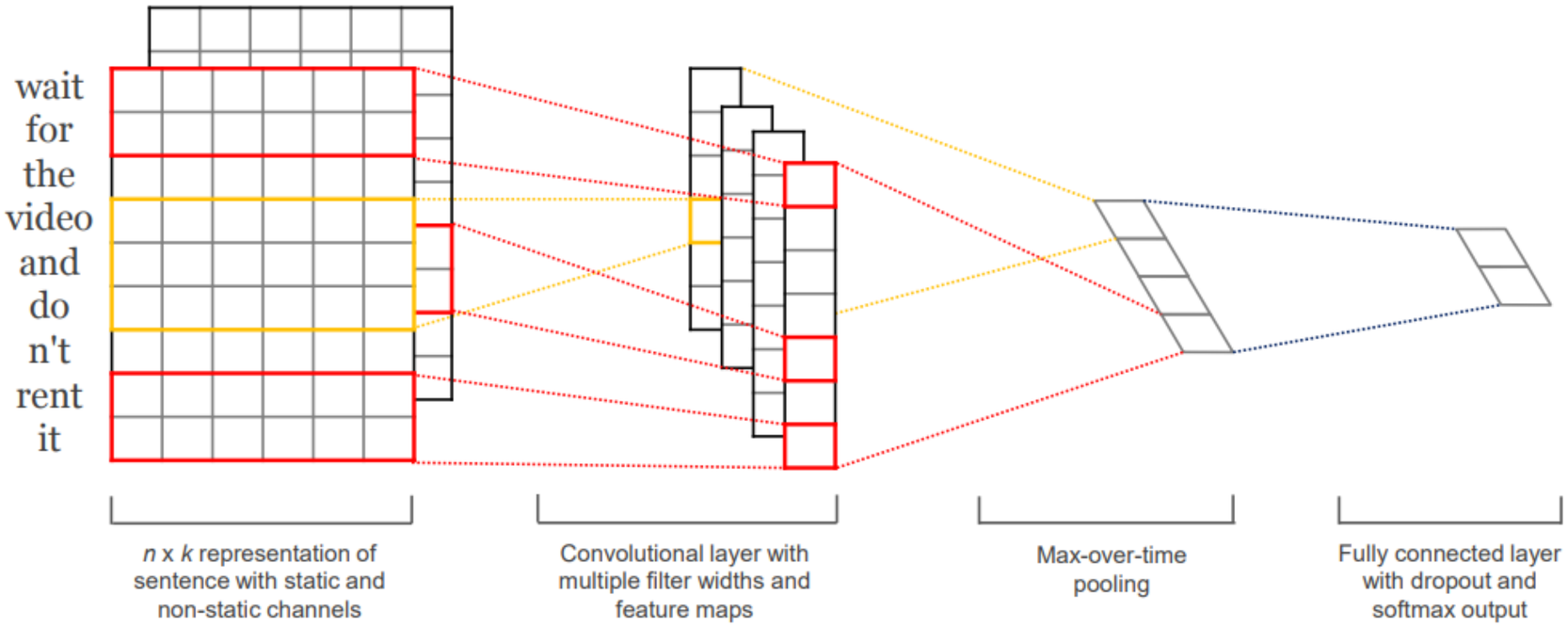


Figure 1: Model architecture with two channels for an example sentence.

Bi-LSTM & CNN

CNN 모델

```python
class CNNClassifier(nn.Module):

    def __init__(
        self,
        input_size,
        word_vec_size,
        n_classes,
        use_batch_norm=False,
        dropout_p=.5,
        window_sizes=[3, 4, 5],
        n_filters=[100, 100, 100],
    ):

        ... # 변수 할당


        super().__init__()

        self.emb = nn.Embedding(input_size, word_vec_size)
        # Use nn.ModuleList to register each sub-modules.
        self.feature_extractors = nn.ModuleList()
        for window_size, n_filter in zip(window_sizes, n_filters):
            self.feature_extractors.append(
                nn.Sequential(
                    nn.Conv2d(
                        in_channels=1, # We only use one embedding layer.
                        out_channels=n_filter,
                        kernel_size=(window_size, word_vec_size),
                    ),
                    nn.ReLU(),
                    nn.BatchNorm2d(n_filter) if use_batch_norm else nn.Dropout(dropout_p),
                )
            )

        # An input of generator layer is max values from each filter.
        self.generator = nn.Linear(sum(n_filters), n_classes)
        # We use LogSoftmax + NLLLoss instead of Softmax + CrossEntropy
        self.activation = nn.LoSoftmax(dim=-1)[])
}
```

앙상블 작동 방식

```python
...

        y_hats = []
        # Get prediction with iteration on ensemble.
        for model in ensemble:
            if config.gpu_id >= 0:
                model.cuda(config.gpu_id)
            # Don't forget turn-on evaluation mode.
            model.eval()

            y_hat = []
            for idx in range(0, len(lines), config.batch_size):
                # Converts string to list of index.
                x = text_field.numericalize(
                    text_field.pad(lines[idx : idx + config.batch_size]),
                    device="cuda:%d" % config.gpu_id if config.gpu_id >= 0 else "cpu",
                )

                y_hat += [model(x).cpu()]
            # Concatenate the mini-batch wise result
            y_hat = torch.cat(y_hat, dim=0)
            # |y_hat| = (len(lines), n_classes)

            y_hats += [y_hat]

            model.cpu()
        # Merge to one tensor for ensemble result and make probability from log-prob.
        y_hats = torch.stack(y_hats).exp()
        # |y_hats| = (len(ensemble), len(lines), n_classes)
        y_hats = y_hats.sum(dim=0) / len(ensemble)  # Get average
        # |y_hats| = (len(lines), n_classes)


...
```

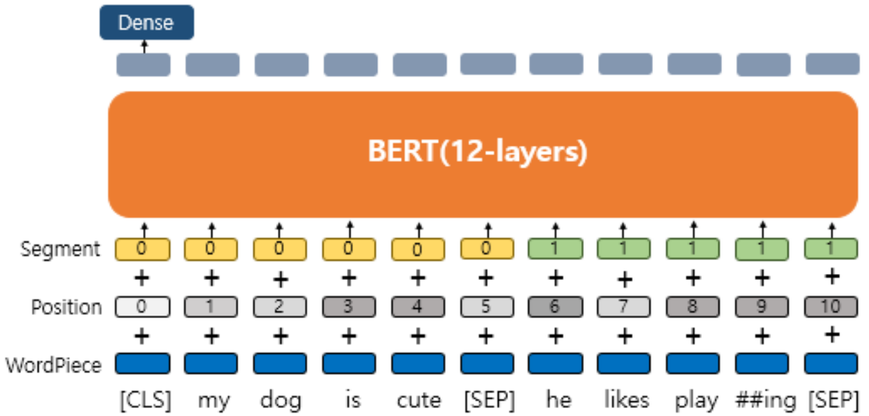
Bi-LSTM & CNN

성능 : 생각보다 높지 않음

		SCORE		기본					RNN			CNN			
코드명	VALIDATION	PRIVATE	PUBLIC	전처리 코드	batch size	n epochs	word vec size	dropout	rnn	hidden size	n layers	cnn	batch norm	window sizes	n filters
R1	0.8237			P1	128	10	256	0.3	0	512	4	X	X	X	X
		0.784932107	0.814895947	P2(약간다른)	128	10	256	0.3	0	512	4	X	X	X	X
RC1	0.7794			P2	128	10	256	0.3	0	512	4	0	X	3 4 5 6 7 8	128 128 128 128 128 128
RC2	0.8323 / 0.7919			P2	128	10	256	0.3	0	512	6	0	x	3 4 5 6 7 8	128 128 128 128 128 128
RC3	0.8164 / 0.7879			P2	128	10	256	0.3	0	512	10	0	x	3 4 5 6 7 8	128 128 128 128 128 128
RC4	0.8219 / 0.7960			P2	128	10	128	0.3	0	1024	6	0	x	3 4 5 6 7 8	128 128 128 128 128 128
RC5	0.8318 / 0.7947	0.714848883	0.751807229	P2	128	10	256	X	0	1024	6	0	0	3 4 5 6 7 8	128 128 128 128 128 128

BERT

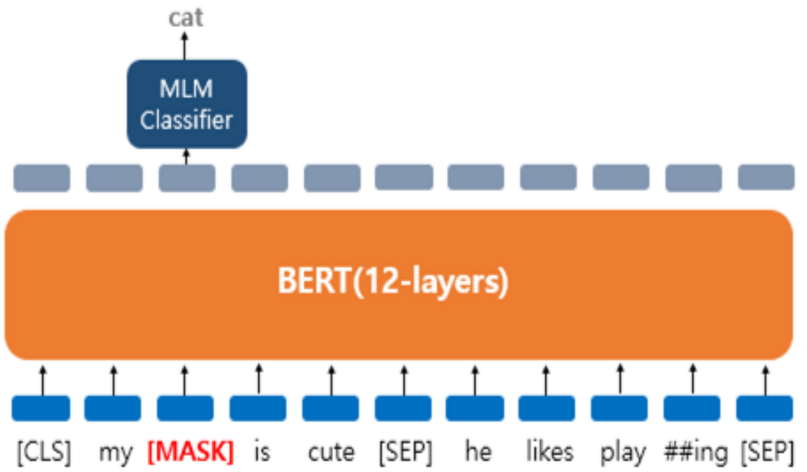
- BERT는 Transformer를 이용해 구현
- BERT Base와 BERT Large는 레이어의 개수에 따라 구분
- BERT의 model pretrain 방식
  - MLM (Masked Language Model)
  - NSP (Next Sentence Prediction)
- BERT의 임베딩
  - **Contextual Embedding** : 실질적인 입력이 되는 워드 임베딩
  - **Position Embedding** : 위치 정보를 학습하기 위한 임베딩
  - **Segment Embedding** : 두 개의 문장을 구분하기 위한 임베딩



BERT

Masked Language Model(MLM)

- Bert의 사전 훈련을 위해, 인공 신경망의 입력으로 들어가는 입력 테스트의 15%의 단어를 랜덤으로 마스킹
- 인공 신경망에게 가려진 단어들을 예측하도록 함 ⇒ 중간에 단어들에 구멍을 뚫어놓고, 구멍에 들어갈 단어들을 예측
- EX) '나는 [MASK]에 가서 그곳에서 빵과 [MASK]를 샀다' → 슈퍼, 우유 맞추기



- ‘dog’ 토큰은 [MASK]로 변경되었습니다.
- ‘he’는 랜덤 단어 ‘king’으로 변경되었습니다.
- ‘play’는 변경되진 않았지만 예측에 사용됩니다.

BERT

Next Sentence Prediction(NSP)

- Bert는 두개의 문장을 준 후에 이 문장이 이어지는 문장인지 아닌지를 맞추는 방식으로 훈련
- 50 : 50 비율로 실제 이어지는 두 개의 문장과 랜덤으로 이어 붙인 두 개의 문장을 주고 훈련 시킴
- Sentence A와 Sentence B라고 했을 때, 문장의 연속성을 확인한 경우와 그렇지 않은 경우를 보여줌

- 이어지는 문장의 경우
  - Sentence A : The man went to the store.
  - Sentence B : He bought a gallon of milk.

  Label = IsNextSentence

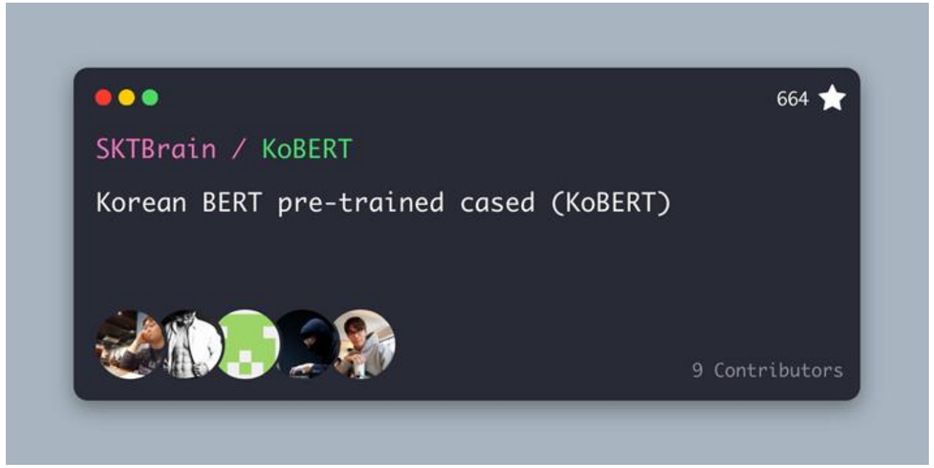
- 이어지는 문장이 아닌 경우
  - Sentence A : Thte man went to the store
  - Sentence B : dogs are so cute.

  Label = NotNextSentence.

- 80%의 단어들은 [MASK]로 변경
  Ex) The man went to the store → The man went to the [MASK]
- 10%의 단어들은 랜덤으로 단어가 변경
  Ex) The man went to the store → The man went to the dog
- 10%의 단어들은 동일하게 둠
  Ex) The man went to the store → The man went to the store

KoBERT

- 구글 BERT base multilingual cased의 한국어 성능 한계로 인해 모델 개발
- 한글 위키 기반(문장 5M, 단어 54M)으로 학습한 토크나이저 (SentencePiece)



KoBERT(1)

1) 입력 데이터 만들기

```
# train, valid set 분리
from sklearn.model_selection import train_test_split
train,valid = train_test_split(train,test_size=0.2, random_state=42)
print('train shape is:',len(train))
print('valid shape is:',len(valid))
```

```
. KoBERT 입력 데이터로 만들기
```

```
# 토크나이저 선언
tokenizer = get_tokenizer()
tok = nlp.data.BERTSPTokenizer(tokenizer,vocab,lower=False)
```

```
# 모델 초기화 -> BERT 모델, Vocabulary 불러오기
bertmodel,vocab = get_pytorch_kobert_model()
```

```
class BERTDataset(Dataset):
  def __init__(self,dataset, sent_idx, label_idx, bert_tokenizer, max_len, pad, pair):
    transform = nlp.data.BERTSentenceTransform(bert_tokenizer, max_seq_length = max_len, pad= pad, pair = pair)

    self.sentences = [transform([i[sent_idx]])for i in dataset]
    self.labels = [np.int32(i[label_idx]) for i in dataset]

  def __getitem__(self,i):
    return (self.sentences[i] + (self.labels[i],))

  def __len__(self):
    return (len(self.labels))
```

```
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6}
```

```
data_train = BERTDataset(train,0,1,tok,max_len,True,False)
data_valid = BERTDataset(valid,0,1,tok,max_len,True,False)
```

```
using cached model. /content/.cache/kobert_news_wiki_ko_cased-1087f8699e.spiece
```

- SKT Kobert pretrain model 사용
- train, validation 분류
- 토크나이저 선언
- 모델 초기화
- 데이터셋 정의

KoBERT(1)

임베딩 결과 예시

```
data_train[0]
```

```
(array([   2,  665, 5112, 6527, 7953, 7095,  743, 1292, 7003, 4064, 4577,
           3,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
           1,    1,    1,    1,    1,    1,    1,    1,    1], dtype=int32),
 array(12, dtype=int32),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       dtype=int32),
 1)
```

- 첫 번째 : 패딩된 시퀀스
- 두 번째 : 길이와 타입에 관한 내용
- 세 번째 : 어텐션 마스크 시퀀스

KoBERT(1)

2) 학습 모델 생성

```
# KoBERT 학습모델 만들기

class BERTClassifier(nn.Module):
  def __init__(self,
               bert,
               hidden_size = 768, #
               num_classes = 7, # softmax 사용 <- binary일 경우는 2
               dr_rate = None,
               params = None):
    super(BERTClassifier,self).__init__()
    self.bert = bert
    self.dr_rate = dr_rate #

    self.classifier = nn.Linear(hidden_size,num_classes)
    if dr_rate:
      self.dropout = nn.Dropout(p=dr_rate)

  def gen_attention_mask(self, token_ids, valid_length):
    attention_mask = torch.zeros_like(token_ids)
    for i,v in enumerate(valid_length):
      attention_mask[i][:v] = 1
    return attention_mask.float()

  def forward(self, token_ids, valid_length, segment_ids):
    attention_mask = self.gen_attention_mask(token_ids,valid_length)

    _,pooler = self.bert(input_ids = token_ids, token_type_ids = segment_ids.long(),attention_mask = attention_mask.float().to(token_ids.device))
    if self.dr_rate:
      out = self.dropout(pooler)
    return self.classifier(out)

# BERT 모델 불러오기
model = BERTClassifier(bertmodel, dr_rate = 0.5).cuda()
```

multi classification 문제
→ num_class = 7로 설정

KoBERT(1)

3) 모델 학습시키기

```
# optimizer와 schedule 설정
no_decay = ['bias','LayerNorm.weight']
optimizer_grouped_parameters = [
      {'params': [p for n,p in model.named_parameters() if not any(nd in n for nd in no_decay )], 'weight_decay':0.01},
      {'params': [p for n,p in model.named_parameters() if any(nd in n for nd in no_decay)], 'weight_decay':0.0}
]

# 옵티마이저 선언
optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate)
loss_fn = nn.CrossEntropyLoss()

t_total = len(train_dataloader)*num_epochs
warmup_step = int(t_total*warmup_ratio)

scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=warmup_step,num_training_steps=t_total)

# 학습 평가 지표인 accuracy 계산 -> 얼마나 타겟값을 많이 맞추었는가? -> 정확도 측정을 위한 함수 정의
def calc_accuracy(X,Y):
  max_vals, max_indices = torch.max(X,1)
  train_acc = (max_indices == Y).sum().data.cpu().numpy()/max_indices.size()[0]
  return train_acc
```

KoBERT(1)

3) 모델 학습시키기

```
# KoBERT 모델 학습 시키기 -> 모델 학습 시작

for e in range(num_epochs):
  print(f'========================================== epoch %d ==========================================' %(e+1))
  train_acc = 0.0
  test_acc = 0.0
  model.train()
  for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_dataloader)):
    optimizer.zero_grad()
    token_ids = token_ids.long().cuda() ##
    segment_ids = segment_ids.long().cuda()
    valid_length = valid_length
    label = label.long().cuda()
    out = model(token_ids, valid_length, segment_ids)
    loss = loss_fn(out,label)
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm) # grad clipping
    optimizer.step()
    scheduler.step() # update learning rate schedule
    train_acc += calc_accuracy(out, label)
    if batch_id % log_interval == 0 :
      print(f'Iteration %3.d | Train Loss  %.4f | Classifier Accuracy %2.2f' % (batch_id+1, loss.data.cpu().numpy(), train_acc / (batch_id+1)))
  print("epoch {} train acc {}".format(e+1, train_acc / (batch_id+1)))

  model.eval # 평가 모드로 변경

  for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(valid_dataloader)):
    token_ids = token_ids.long().cuda()
    segment_ids = segment_ids.long().cuda()
    valid_length = valid_length
    label = label.long().cuda()
    out = model(token_ids, valid_length, segment_ids)
    test_acc += calc_accuracy(out,label)
  print("epoch {} valid acc {}".format(e+1, test_acc / (batch_id+1)))
```

```
epoch 4 valid acc 0.8897534151894617
========================================== epoch 5 ==========================================
100% 571/571 [11:57<00:00, 1.14s/it]
Iteration   1 | Train Loss  0.0813 | Classifier Accuracy 0.97
Iteration  21 | Train Loss  0.1274 | Classifier Accuracy 0.97
Iteration  41 | Train Loss  0.1459 | Classifier Accuracy 0.97
Iteration  61 | Train Loss  0.0417 | Classifier Accuracy 0.97
Iteration  81 | Train Loss  0.1319 | Classifier Accuracy 0.97
Iteration 101 | Train Loss  0.0225 | Classifier Accuracy 0.97
Iteration 121 | Train Loss  0.1801 | Classifier Accuracy 0.97
Iteration 141 | Train Loss  0.1011 | Classifier Accuracy 0.97
Iteration 161 | Train Loss  0.1408 | Classifier Accuracy 0.97
Iteration 181 | Train Loss  0.0652 | Classifier Accuracy 0.97
Iteration 201 | Train Loss  0.1350 | Classifier Accuracy 0.97
Iteration 221 | Train Loss  0.0335 | Classifier Accuracy 0.97
Iteration 241 | Train Loss  0.0664 | Classifier Accuracy 0.97
Iteration 261 | Train Loss  0.0470 | Classifier Accuracy 0.97
Iteration 281 | Train Loss  0.0401 | Classifier Accuracy 0.97
Iteration 301 | Train Loss  0.0384 | Classifier Accuracy 0.97
Iteration 321 | Train Loss  0.0367 | Classifier Accuracy 0.97
Iteration 341 | Train Loss  0.0603 | Classifier Accuracy 0.97
Iteration 361 | Train Loss  0.0639 | Classifier Accuracy 0.97
Iteration 381 | Train Loss  0.0545 | Classifier Accuracy 0.97
Iteration 401 | Train Loss  0.0150 | Classifier Accuracy 0.97
Iteration 421 | Train Loss  0.0321 | Classifier Accuracy 0.97
Iteration 441 | Train Loss  0.0776 | Classifier Accuracy 0.97
Iteration 461 | Train Loss  0.1029 | Classifier Accuracy 0.97
Iteration 481 | Train Loss  0.0357 | Classifier Accuracy 0.97
Iteration 501 | Train Loss  0.0439 | Classifier Accuracy 0.97
Iteration 521 | Train Loss  0.0565 | Classifier Accuracy 0.97
Iteration 541 | Train Loss  0.0462 | Classifier Accuracy 0.97
Iteration 561 | Train Loss  0.0171 | Classifier Accuracy 0.97
epoch 5 train acc 0.9713222416812609
100% 143/143 [01:08<00:00, 2.28it/s]
epoch 5 valid acc 0.8892604488534721
```

KoBERT(1)

4) 추론

```
data_test = BERTDataset(test,0,1,tok,max_len,True,False) # test dataset에 대해서 토큰화, 패딩 등을 진행
test_dataloader = torch.utils.data.DataLoader(data_test, batch_size = batch_size, num_workers = 5) # torch 형식의 dataset 만들어줌
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 5 worker process
  cpuset_checked))
```

```
predictions = []
with torch.no_grad(): # 수동으로 변화도 버퍼를 0으로 설정하는 것에 유의 -> 변화도가 누적되기 때문
  for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(test_dataloader)):
    token_ids = token_ids.long().cuda()
    segment_ids = segment_ids.long().cuda()
    valid_length = valid_length
    label = label.long().cuda()
    out = model(token_ids, valid_length, segment_ids)
    _, max_indices = torch.max(out,1)

    predictions.extend(max_indices.squeeze(0).detach().cpu().numpy()) # squeeze(0) : 1차원을 제거해줌
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  This is separate from the ipykernel package so we can avoid doing imports until
100% 143/143 [01:10<00:00, 2.28it/s]
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 5 worker process
  cpuset_checked))
```

```
print(predictions)
```

```
[2, 3, 6, 2, 3, 0, 5, 3, 4, 4, 4, 6, 4, 5, 6, 1, 6, 2, 4, 4, 4, 4, 4, 0, 0, 3, 6, 2, 5, 2, 3, 1, 4, 5, 1, 4, 5, 4, 6, 5, 5, 5, 5, 3,
```

Accuracy

- validation accuracy : 97%
- test accuracy : 85%

DACON 제출 결과

kobert_remove_index_0723.csv 0.856516977 0.8258869908

edit

KoBERT(2)

- kykim/bert-kor-base (kykim/bert-kor-base · Hugging Face)
- beomi/kcbert-base (Beomi/KcBERT: 😆 Pretrained BERT model & WordPiece tokenizer trained on Korean Comments)
- beomi/kcbert-large (Beomi/KcBERT: 😆 Pretrained BERT model & WordPiece tokenizer trained on Korean Comments)
  - Out of Memory

KoBERT(2)

성능: kykim/bert-kor-base의 성능이 가장 높았음.

		SCORE		BERT										
코드명	VALIDATION	PRIVATE	PUBLIC	전처리 코드	batch size	n epochs	pretrained model name	use albert	lr	warmup ratio	adam epsilon	use radam	valid ratio	max length
FN1	0.8582	0.808147175	0.831544359	P2	128	10	beomi/kcbert-base	X	5.00E-05	0.2	1.00E-08	X	0.2	100
FN2	0.8748	0.818878668	0.849507119	P2	128	10	kykim/bert-kor-base	X	5.00E-05	0.2	1.00E-08	X	0.2	100
FN4	0.8415			P2	128	10	kykim/bert-kor-base	O	5.00E-05	0.2	1.00E-08	X	0.2	100
FN5	0.8596			P2	128	10	kykim/bert-kor-base	X	1.00E-04	0	1.00E-08	O	0.2	100

DistilBERT

Knowledge Distillation : 모델 압축 기술

큰 모델 (teacher - a larger model) / 작은 모델 (student - a compact model)을 두어 학생이 선생의 동작 방식 배우도록 함

→ 선생 모델이 출력하는 확률 분포 (soft target probability)를 배움으로써 자신보다 복잡한 모델들만이 배울 수 있는 signal들을 배우게 됨

- Distillation loss

$$L_{ce} = \sum_i t_i * \log(s_i)$$

- Softmax-temperature

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

DistilBERT

BERT에 Knowledge Distillation 적용한 모델

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	77.6	48.9	84.3	88.6	89.3	89.5	71.3	91.7	91.2	43.7
DistilBERT	76.8	49.1	81.8	90.2	90.2	89.2	62.9	92.7	90.7	44.4

Table 2: **DistilBERT yields to comparable performance on downstream tasks.** Comparison on downstream tasks: IMDb (test accuracy) and SQuAD 1.1 (EM/F1 on dev set). D: with a second step of distillation during fine-tuning.

Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

Table 3: **DistilBERT is significantly smaller while being constantly faster.** Inference time of a full pass of GLUE task STS-B (sentiment analysis) on CPU with a batch size of 1.

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

기존 BERT 모델 대비,

- 사이즈 40% 감소
- 속도 60% 증가
- NLU 능력 97% 유지

DistilBERT

Pretraining

- 기존 kobert의 12 Layer를 3 Layer로 줄임 (distilbert: 6 layer)
- Layer 초기화의 경우 기존 KoBERT의 1, 5, 9번째 layer 값을 그대로 사용
- Pretraining Corpus는 한국어 위키, 나무위키, 뉴스 등 약 10GB의 데이터를 사용했으며, 3 epoch 학습

VS BERT

• DistilBert는 기존의 Bert와 달리 token-type embedding, pooler를 사용하지 않음

<forward return value>

- BertModel :
  sequence_output, pooled_output, (hidden_states), (attentions)
- DistilBert :
  sequence_output, (hidden_states), (attentions)

DistilBERT

Import Tokenizer & Model

```
from kobert_transformers import get_distilkobert_model

_, vocab = get_pytorch_kobert_model()
distilbert_model = get_distilkobert_model()
tokenizer = get_tokenizer()
tok = nlp.data.BERTSPTokenizer(tokenizer, vocab, lower=False)
```

```
/content/mecab-python-0.996/.cache/kobert_v1.zip[                                                  ]
/content/mecab-python-0.996/.cache/kobert_news_wiki_ko_cased-1087f8699e.spiece[
Downloading: 100% 441/441 [00:00<00:00, 11.0kB/s]
Downloading: 100% 108M/108M [00:03<00:00, 40.9MB/s]
Some weights of the model checkpoint at monologg/distilkobert were not used when initializing DistilBertModel: ['voca
- This IS expected if you are initializing DistilBertModel from the checkpoint of a model trained on another task or
- This IS NOT expected if you are initializing DistilBertModel from the checkpoint of a model that you expect to be e
using cached model. /content/mecab-python-0.996/.cache/kobert_news_wiki_ko_cased-1087f8699e.spiece
```

Parameter Setting

```
# Setting parameters - SKT 예시 파라미터 값
max_len = 48 # 해당 길이를 초과하는 단어에 대해선 bert가 학습 x (max length 44)
batch_size = 32
warmup_ratio = 0.1
num_epochs = 5
max_grad_norm = 1
log_interval = 20
learning_rate = 5e-5
```

Split Dataset

```
train = train[['cleaned_title', 'topic_idx']]
test = test[['cleaned_title']]
```

```
# train, valid set 분리
from sklearn.model_selection import train_test_split
train, valid = train_test_split(train,test_size=0.2, random_state=42)
```

```
print('train shape is:',len(train))
print('valid shape is:',len(valid))
```

```
train shape is: 36523
valid shape is: 9131
```

DistilKoBERT

BERT Dataset&Classifier

```
class BERTDataset(Dataset):
    def __init__(self, dataset, sent_idx, label_idx, bert_tokenizer, max_len,
                 pad, pair):
        data_list = [list(dataset.iloc[i]) for i in range(len(dataset))]
        transform = nlp.data.BERTSentenceTransform(
            bert_tokenizer, max_seq_length=max_len, pad=pad, pair=pair)

        self.sentences = [transform([i[sent_idx]]) for i in data_list]
        self.labels = [np.int64(i[label_idx]) for i in data_list]

    def __getitem__(self, i):
        return (self.sentences[i] + (self.labels[i], ))

    def __len__(self):
        return (len(self.labels))
```

```
data_train = BERTDataset(train,0,1,tok,max_len, True, False)
data_valid = BERTDataset(valid,0,1,tok,max_len, True, False)
```

```
train_dataloader = torch.utils.data.DataLoader(data_train, batch_size=batch_size, num_workers=2)
valid_dataloader = torch.utils.data.DataLoader(data_valid, batch_size=batch_size, num_workers=2)
```

```
class BERTClassifier(nn.Module):
    def __init__(self,
                 bert,
                 hidden_size = 768,
                 num_classes = 7, # softmax 사용 <- binary일 경우는 2
                 dr_rate=None,
                 params=None):
        super(BERTClassifier, self).__init__()
        self.bert = bert
        self.dr_rate = dr_rate

        self.classifier = nn.Linear(hidden_size , num_classes)
        if dr_rate:
            self.dropout = nn.Dropout(p=dr_rate)

    def gen_attention_mask(self, token_ids, valid_length):
        attention_mask = torch.zeros_like(token_ids)
        for i, v in enumerate(valid_length):
            attention_mask[i][:v] = 1
        return attention_mask.float()

    def forward(self, token_ids, valid_length, segment_ids):
        attention_mask = self.gen_attention_mask(token_ids, valid_length)

        sequence_output = self.bert(input_ids = token_ids,
                                    attention_mask = attention_mask.float().to(token_ids.device))
        if self.dr_rate:
          # distilbert [CLS] 토큰 sequence_output[0][:, 0]
          out = self.dropout(sequence_output[0][:, 0])
        return self.classifier(out)
```

DistilKoBERT

Optimizer (AdamW) & Schedule Setting

```
# distilbert_model train & validate
# Prepare optimizer and schedule (linear warmup and decay)
no_decay = ['bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {'params': [p for n, p in distilbert_model.named_parameters() if not any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in distilbert_model.named_parameters() if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}
]

optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate)
loss_fn = nn.CrossEntropyLoss()

t_total = len(train_dataloader) * num_epochs
warmup_step = int(t_total * warmup_ratio)

scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=warmup_step, num_training_steps=t_total)
```

Model Training

```
epoch 1 batch id 1101 loss 0.565188467502594 train acc 0.7194311989100818
epoch 1 batch id 1121 loss 0.3452472388744354 train acc 0.7217885816235504
epoch 1 batch id 1141 loss 0.5830373167991638 train acc 0.723542944785276
epoch 1 train acc 0.723705421111288
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:40: TqdmDeprecati
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
100% 286/286 [05:23<00:00, 1.08it/s]
epoch 1 validation acc 0.8578750794659885
100% 1142/1142 [1:07:42<00:00, 2.92s/it]
epoch 5 batch id 1061 loss 0.4031526446342468 train acc 0.9245110744580585
epoch 5 batch id 1081 loss 0.2723281979560852 train acc 0.924404486586494
epoch 5 batch id 1101 loss 0.13583002984523773 train acc 0.9243869209809265
epoch 5 batch id 1121 loss 0.1264110505580902 train acc 0.9241748438893844
epoch 5 batch id 1141 loss 0.36717286705970764 train acc 0.9239975898334793
epoch 5 train acc 0.9240641418563923
100% 286/286 [05:19<00:00, 1.12it/s]
epoch 5 validation acc 0.8662885410044501
```

DistilKoBERT

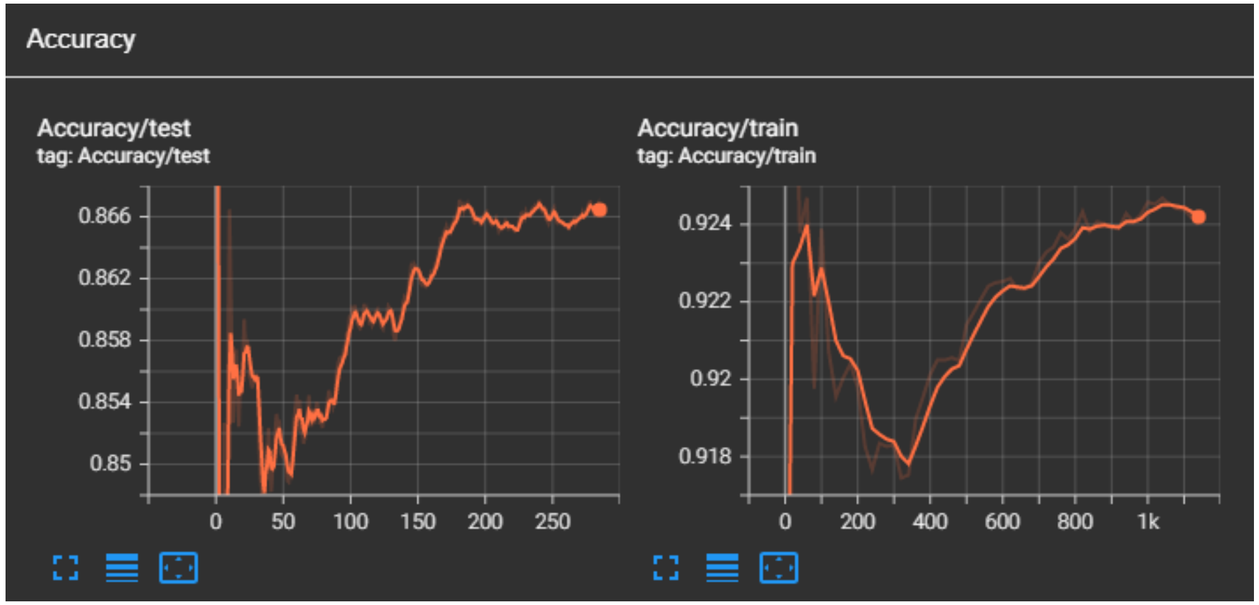
모델 학습 과정 (5 epochs)

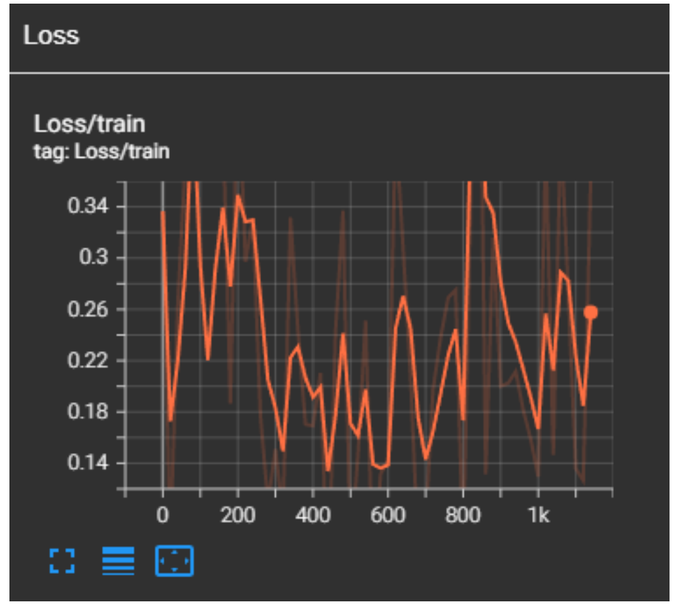
Tensorboard summarywriter로
학습과정 기록

```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter()
running_loss = 0.0

for e in range(num_epochs):
    train_acc = 0.0
    test_acc = 0.0
    model.train()

    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_dataloader)):
        optimizer.zero_grad()
        token_ids = token_ids.long()
        segment_ids = segment_ids.long()
        valid_length = valid_length
        label = label.long()

        out = model(token_ids, valid_length, segment_ids)

        loss = loss_fn(out, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
        optimizer.step()
        scheduler.step()  # Update learning rate schedule
        running_loss += loss.item()

        train_acc += calc_accuracy(out, label)

        if batch_id % log_interval == 0:
            print("epoch {} batch id {} loss {} train acc {}".format(e+1, batch_id+1, loss.data.cpu().numpy(), train_acc / (batch_id+1)))
            writer.add_scalar("Loss/train", loss.data.cpu().numpy(), batch_id)
            writer.add_scalar("Accuracy/train", train_acc / (batch_id+1), batch_id)

    print("epoch {} train acc {}".format(e+1, train_acc / (batch_id+1)))


    model.eval()

    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(valid_dataloader)):
        token_ids = token_ids.long()
        segment_ids = segment_ids.long()
        valid_length= valid_length
        label = label.long()
        out = model(token_ids, valid_length, segment_ids)
        test_acc += calc_accuracy(out, label)
        writer.add_scalar("Accuracy/test", test_acc / (batch_id+1), batch_id)

    print("epoch {} validation acc {}".format(e+1, test_acc / (batch_id+1)))
```

DistilKoBERT

학습 과정 기록 시각화





DistilKoBERT

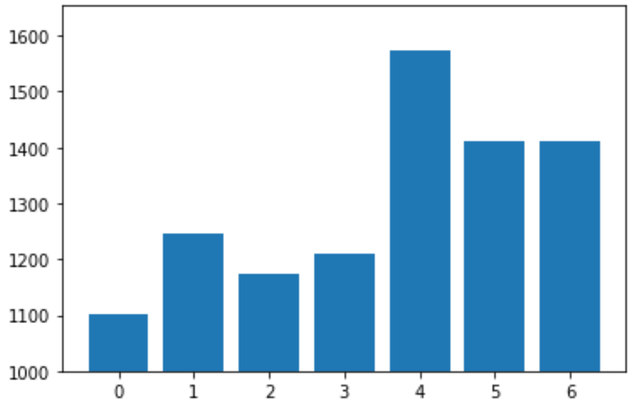
Test data output

```
submission['topic_idx'] = label_answer
submission
```

	index	topic_idx
0	45654	0
1	45655	3
2	45656	2
3	45657	2
4	45658	3
...	...	...
9126	54780	3
9127	54781	2
9128	54782	2
9129	54783	0
9130	54784	6

9131 rows × 2 columns

결과 인덱스 분포



DACON 제출 최종 점수

최신순 점수순

	제목	제출 일시	public점수 private점수	제출선택
644944	distilbert_result.csv edit	2022-02-01 08:57:56	0.8332968237 0.8096802453	☑

결과 비교표

	SimpleDNN	RNN	Bi-LSTM	Bi-LSTM(with FastText)	Bi-LSTM & CNN	KoBERT(1)	KoBERT(2)	DistilBERT
private	0.7568	0.7520	0.7678	0.7380	0.7849	0.8258	0.8188	0.8096
public	0.7877	0.7886	0.7875	0.7805	0.8148	0.8565	0.8495	0.8332

* DACON 제출 스코어 (최고 기록)

KoBERT (1) 모델의 성능이 가장 높게 나타남

의의 및 한계점

의의

- 배웠던 NLP 지식을 활용해 전처리 ~ 모델 생성까지 직접 시도해볼 수 있었음
- 여러 가지 딥러닝 모델을 구현
- 관련된 모델 논문 스터디 진행

한계

- [BERT] 시간/메모리 상의 한계로 다양한 실험 해보지 못함
- 여러 가지 모델 앙상블 시도

LSTM

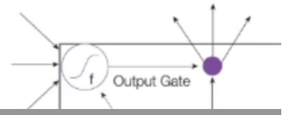
LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 인풋게이트
- 포겟 게이트
- 블락 인풋 게이트
- 셀
- 아웃풋 게이트

▸ LSTM(장단기 기억 네트워크)와 경사도

이제 인풋 게이트, 포겟 게이트, 아웃풋 게이트에서 일어나는 연

• 그림은 하나의 메모리 블럭을 나타냅니다. (인풋 게이트, 아웃
게이트, 셀)



1. 기본 개념

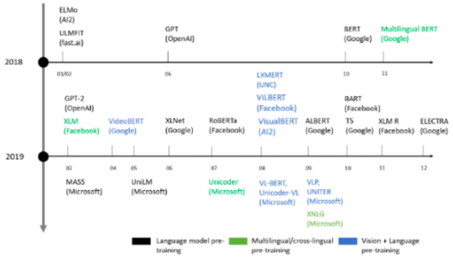
BERT(Bidirectional Encoder Representations from Transformers)는 2018년에 구글이 공개한 사전 훈련된 모델

질문에 대한 대답, 텍스트 생성 등과 같은 태스크에서 가장 좋은 성능을 도출해 자연어 처리 분야에 크게 기여

워드투벡터와 같은 context-free 임베딩 모델을 사용하면 동일한 단어가 쓰였을 때 임베딩 벡터가 동**일하게 표현**

→ 문맥을 고려하지 못해 다의어나 동음이의어를 구분하지 못하는 문제점이 있음

• Pretraining and finetuning (Transfer learning) with Big-LM.



BERT는 모든 단어의 문맥상 의미를 이해하기 위해 **문장의 각 단어를 문장의 다른 모든 단어와 연결**시켜 이해한다. (셀프 어텐션 활용)

> A : He got bit by Python (비단뱀) 'Python' - 'bit'단어의 강한 연결 관계를 파악

2. 동작 방식

BERT는 트랜스포머 모델을 기반으로 하며, 그 중 **인코더**만 사용

▼ 동작 방식 예시

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

Victor SANH, Lysandre DEBUT, Julien CHAUMOND, Thomas WOLF
Hugging Face
{victor,lysandre,julien,thomas}@huggingface.co

Convolutional Neural Networks for Sentence Classification

Yoon Kim
New York University
yhk255@nyu.edu

Abstract

We report on a series of experiments with convolutional neural networks (CNN) trained on top of pre-trained word vectors for sentence-level classification tasks. We show that a simple CNN with little hyperparameter tuning and static vectors achieves excellent results on multiple benchmarks. Learning task-specific vectors through fine-tuning offers further gains in performance. We additionally propose a simple modification to the architecture to allow for the use of both task-specific and static vectors. The CNN models discussed herein improve upon the state of the art on 4 out of 7 tasks, which include sentiment analysis and question classification.

Enriching Word Vectors with Subword Information

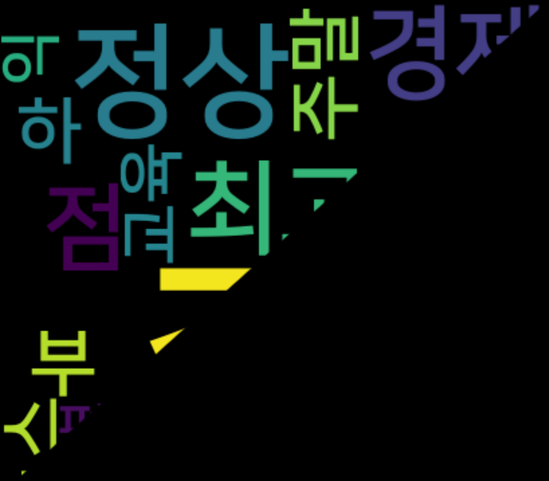
Piotr Bojanowski* and **Edouard Grave*** and **Armand Joulin** and **Tomas Mikolov**
Facebook AI Research
{bojanowski,egrave,ajoulin,tmikolov}@fb.com

Abstract

Continuous word representations, trained on large unlabeled corpora are useful for many natural language processing tasks. Popular models that learn such representations ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large vocabularies and many rare words. In this paper, we propose a new approach based on the skipgram model, where each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram; words being represented as the sum of these representations. Our method is fast, allowing to train models on large corpora quickly and allows us to compute word representations for words that did not appear in the training data. We evaluate our word representations on nine different languages, both on word similarity and analogy tasks. By comparing to recently proposed morphological word representations, we show that our vectors achieve state-of-the-art performance on these tasks.

et al., 2010; Baroni and Lenci, 2010). In the neural network community, Collobert and Weston (2008) proposed to learn word embeddings using a feed-forward neural network, by predicting a word based on the two words on the left and two words on the right. More recently, Mikolov et al. (2013b) proposed simple log-bilinear models to learn continuous representations of words on very large corpora efficiently.

Most of these techniques represent each word of the vocabulary by a distinct vector, without parameter sharing. In particular, they ignore the internal structure of words, which is an important limitation for morphologically rich languages, such as Turkish or Finnish. For example, in French or Spanish, most verbs have more than forty different inflected forms, while the Finnish language has fifteen cases for nouns. These languages contain many word forms that occur rarely (or not at all) in the training corpus, making it difficult to learn good word representations. Because many word formations follow



thank you!