
Text Representation

Practical Exercise

확인 문제

- 다음 두 단어의 공기 정보 기반 유사도를 계산하십시오.
 - [조건]
 - 공기 빈도수는 1로 가정
 - 유사도 계산식은 Cosine Similarity 사용
 - [단어]
 - 코로나: 감염, 목욕탕, 바이러스, 신종, 생활, 일상
 - 메르스: 감염, 바이러스, 발생, 지금, 환자



사전 만들기

Mecab 인스톨

```
!git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git
%cd Mecab-ko-for-Google-Colab
!bash install_mecab-ko_on_colab190912.sh
```

실행

```
texts = ["죽는 날까지 하늘을 우러러 한 점 부끄럼이 없기를",
        "잎새에 이는 바람에도 나는 괴로워 했다.",
        "별을 노래하는 마음으로 모든 죽어 가는 것을 사랑해야지",
        "그리고 나한테 주어진 길을 걸어가야겠다.",
        "오늘 밤에도 별이 바람에 스치운다."]
```

```
(indexes, words) = make_dic(texts)
```

```
print(word2index(indexes, '하늘'), word2index(indexes, '학교'))
print(index2word(words, 4), index2word(words, 0))
```

```
사전 크기: 24
4 0
하늘 unk
```



사전 만들기

make_dic 함수

```
from konlpy.tag import Mecab
from nltk import FreqDist
import numpy as np

def make_dic(texts):

    words = []
    m = Mecab()

    for snt in texts:
        res = m.pos(snt)
        for (lex,pos) in res:
            if pos[0] == 'N' or pos[0] == 'V':
                words.append(lex)

    # 단어의 빈도수 계산
    dic = FreqDist(np.hstack(words))
    print('사전 크기: {0:d}'.format(len(dic)))
```

```
# 인덱스를 저장할 변수 초기화
indexes = {}
words = {}

# 단어에 고유 번호(인덱스) 부여
for num, word in enumerate(dic):
    idx = num+2
    indexes[word] = idx
    words[idx] = word

# 문장 길이 정규화에 사용한 패딩 인덱스
indexes['pad'] = 1
words[1] = 'pad'
# 사전에 없는 단어에 대한 인덱스
indexes['unk'] = 0
words[0] = 'unk'

return (indexes, words)
```

```
def word2index(indexes, word):
    idx = indexes[word] if word in indexes else indexes['unk']
    return idx

def index2word(words, index):
    w = words[index] if index in words else -1
    return w
```



Word2Vec 사용하기

학습 데이터 만들기

```
from konlpy.tag import Mecab
import numpy as np
from gensim.models import Word2Vec, KeyedVectors
```

```
texts = ["죽는 날까지 하늘을 우러러 한 점 부끄럼이 없기를",
        "잎새에 이는 바람에도 나는 괴로워 했다.",
        "별을 노래하는 마음으로 모든 죽어 가는 것을 사랑해야지",
        "그리고 나한테 주어진 길을 걸어가야겠다.",
        "오늘 밤에도 별이 바람에 스치운다."]
```

Word2Vec 학습에 사용할 데이터 만들기

```
m = Mecab()
result = []

for sent in texts:
    tag = m.pos(sent)
    words = []
    for (lex,pos) in tag:
        if pos[0] == 'N':
            words.append(lex)
    result.append(words)

print(result, "\n")
```

대용량의
말뭉치를
이용

명사류(체언)만
추출

```
[['날', '하늘', '점', '부끄럼'], ['잎새', '바람', '나'],
 ['별', '노래', '마음', '것', '사랑'], ['나', '길'], ['밤', '별', '바람']]
```



Word2Vec 사용하기

학습 및 평가

Word2Vec 학습시키기

```
model = Word2Vec(sentences=result, size=10, window=1, min_count=1, workers=1, sg=0)
```

값 읽어오기

```
print(model.wv['하늘'])
```

유사한 단어 가져오기

```
print(model.wv.most_similar("하늘"), "\n")
```

- size: 임베딩 차원
- window: 컨텍스트 윈도우 크기
- min_count: 단어 최소 빈도 수 제한
- workers: 학습을 위한 프로세스 수
- sg: 0 (CBOW), 1 (Skip-gram)

Word2Vec 모델 저장하기

```
model.wv.save_word2vec_format('/gdrive/My Drive/colab/text_rep/test_w2v')
```

Word2Vec 모델 로드하기

```
loaded_model = KeyedVectors.load_word2vec_format("/gdrive/My Drive/colab/text_rep/test_w2v")
```

값 읽어오기

```
print(loaded_model.wv['하늘'])
```

유사한 단어 가져오기

```
print(loaded_model.wv.most_similar("하늘"))
```

```
[-0.00181364  0.01349637  0.0494625  -0.04351401 -0.0074907  -0.00855643
 0.01056736 -0.04086038  0.0170731   0.01116189]
[('날', 0.2550535202026367), ('사랑', 0.19656619429588318), ('바람', 0.184

[-0.00181364  0.01349637  0.0494625  -0.04351401 -0.0074907  -0.00855643
 0.01056736 -0.04086038  0.0170731   0.01116189]
[('날', 0.2550535202026367), ('사랑', 0.19656619429588318), ('바람', 0.184
```



Pre-trained Word2Vec

- 영어
 - 구글이 학습한 300만 단어 벡터
 - <https://drive.google.com/file/d/0B7XkCwpl5KDYNINUTTISS21pQmM/edit>
- 다국어(한국어 포함)
 - <https://github.com/Kyubyong/wordvectors>

Language	ISO 639-1	Vector Size	Corpus Size	Vocabulary Size
Bengali (w) Bengali (f)	bn	300	147M	10059
Catalan (w) Catalan (f)	ca	300	967M	50013
Chinese (w) Chinese (f)	zh	300	1G	50101
Danish (w) Danish (f)	da	300	295M	30134
Korean (w) Korean (f)	ko	200	339M	30185



GloVe 사용하기

GloVe 인스톨

```
# colab 환경에 glove 설치
! pip install glove-python-binary
```

학습 및 평가

```
# Co-Occurrence Matrix 생성
corpus = Corpus()
corpus.fit(result, window=1)
```

```
# GloVe 학습시키기
glove = GloVe(no_components=10, learning_rate=0.05)
glove.fit(corpus.matrix, epochs=20, no_threads=1, verbose=True)
glove.add_dictionary(corpus.dictionary)
```

```
# 값 읽어오기
print(glove.word_vectors[glove.dictionary['하늘']])
# 유사한 단어 가져오기
print(glove.most_similar("하늘"), "\n")
```

```
# GloVe 모델 저장하기
glove.save('/gdrive/My Drive/colab/text_rep/test_glove')

# GloVe 모델 로드하기
loaded_model = glove.load("/gdrive/My Drive/colab/text_rep/test_glove")

# 값 읽어오기
print(loaded_model.word_vectors[glove.dictionary['하늘']])
# 유사한 단어 가져오기
print(loaded_model.most_similar("하늘"))
```

10차원 임베딩

```
[ 0.02104036 -0.01404578 -0.04851466  0.01803728 -0.0152335
 -0.04581227 -0.04701197 -0.00781063 -0.01010458]
[('부끄럼', 0.6554835514510583), ('점', 0.5460033054824257)]

[ 0.02104036 -0.01404578 -0.04851466  0.01803728 -0.0152335
 -0.04581227 -0.04701197 -0.00781063 -0.01010458]
[('부끄럼', 0.6554835514510583), ('점', 0.5460033054824257)]
```



Embedding Layer

- 임베딩 벡터를 사용하는 방법
 - 사전 학습된 벡터 사용
 - Word2Vec, GloVe 등을 읽어와서 사용하며 학습 과정에 업데이트 없음
 - 학습 과정에서 벡터 업데이트
 - 사전 학습된 벡터 또는 임의의 벡터를 만들고 학습 과정에서 업데이트 → 파인 튜닝 (fine-tuning)
 - 파이토치에서는 `nn.Embedding()`을 사용하여 구현

Word → Integer → lookup Table → Embedding vector



사전 만들기과 동일
(단, Tensor 형태)

그림 출처: <https://wikidocs.net/64779>



Embedding Layer

사전 만들기

```
import torch
import torch.nn as nn

train_data = 'I like deep learning I like NLP I enjoy flying'

# 중복을 제거한 단어들의 집합인 단어 집합 생성.
word_set = set(train_data.split())

# 단어 집합의 각 단어에 고유한 정수 맵핑
vocab = {word: i+2 for i, word in enumerate(word_set)}
vocab['unk'] = 0
vocab['pad'] = 1
print(vocab)

# 임베딩 테이블: (단어수, 임베딩 사이즈)
embedding_table = torch.FloatTensor([
    [ 0.0, 0.0, 0.0],
    [ 0.0, 0.0, 0.0],
    [ 0.1, 0.8, 0.3],
    [ 0.7, 0.8, 0.2],
    [ 0.1, 0.8, 0.7],
    [ 0.9, 0.2, 0.1],
    [ 0.1, 0.1, 0.9],
    [ 0.2, 0.1, 0.7],
    [ 0.3, 0.1, 0.1]])
```

미등록어와 패딩을
위한 인덱스 부여

Word2Vec, GloVe,
Random, ...

```
{'deep': 2, 'I': 3, 'flying': 4, 'learning': 5,
 'NLP': 6, 'enjoy': 7, 'like': 8, 'unk': 0, 'pad': 1}
```



Embedding Layer

임베딩 층 만들기

```
# 입력 문장
input_snt = 'I like football'.split()

# 각 단어를 정수로 변환
idxes=[]

for word in input_snt:
    idx = vocab[word] if word in vocab else vocab['unk']
    idxes.append(idx)

idxes = torch.LongTensor(idxes)

# 입력 문장의 임베딩 가져오기: ['I', 'like', 'football (unk)']
lookup_result = embedding_table[idxes, :]
print(lookup_result)

# 임베딩 층 만들기
embedding_layer = nn.Embedding(num_embeddings=len(vocab), embedding_dim=3, padding_idx=1)
print(embedding_layer.weight)
```

```
tensor([[0.7000, 0.8000, 0.2000],
        [0.3000, 0.1000, 0.1000],
        [0.0000, 0.0000, 0.0000]])
Parameter containing:
tensor([[ 0.4472, -0.6269,  1.7040],
        [ 0.0000,  0.0000,  0.0000],
        [ 1.5586,  0.9990, -0.3825],
        [-0.9490,  0.0322,  1.6749],
        [-2.0015, -0.1524, -0.7792],
        [ 0.8227,  1.0647,  1.2288],
        [-1.6015, -1.4856,  0.2005],
        [ 0.5156,  0.1298, -1.1945],
        [-0.5552,  0.2856, -1.5769]], requires_grad=True)
```

추가할 인자:
embeddings=embedding_table,
freeze=False



과제: Word2Vec

- 한국어 Word2Vec을 이용하여 아래에 주어진 단어들을 벡터 연산한 후에 가장 가까운 상위 3개의 결과를 제시 하시오. (10점)
 - Word2Vec
 - <https://github.com/Kyubyong/wordvectors>
 - 벡터 연산
 - 한국-서울+파리 → 프랑스, 영국, 미국
 - 컴퓨터+공학+인문학
 - 사랑+이별 → 슬픔, 추억, 절망
 - 충청북도-청주+강원도
 - 인간-사랑
 - 미술-물감+음악



Pretrained Language Model: PART I

Practical Exercise

실습: Sentiment Analysis

- 대용량 언어 모델인 BERT를 활용하여 입력된 문장의 긍/부정을 분류하는 감성 분석 모델을 만드시오.

- 예제

- 아 더빙... 진짜 짜증나네요 목소리 → Negative
- 정말재미있고 교훈적인 영화이네요! → Positive

- 모델 입력


- 한글 문장

- 모델 출력: N or P

- N: Negative, P: Positive

- 데이터 형식

- 워드피스 토큰 열 `wt` 레이블
- 예제: `_아 _더 빙 . . _진짜 _짜 증 나 네요 _목소리` negative

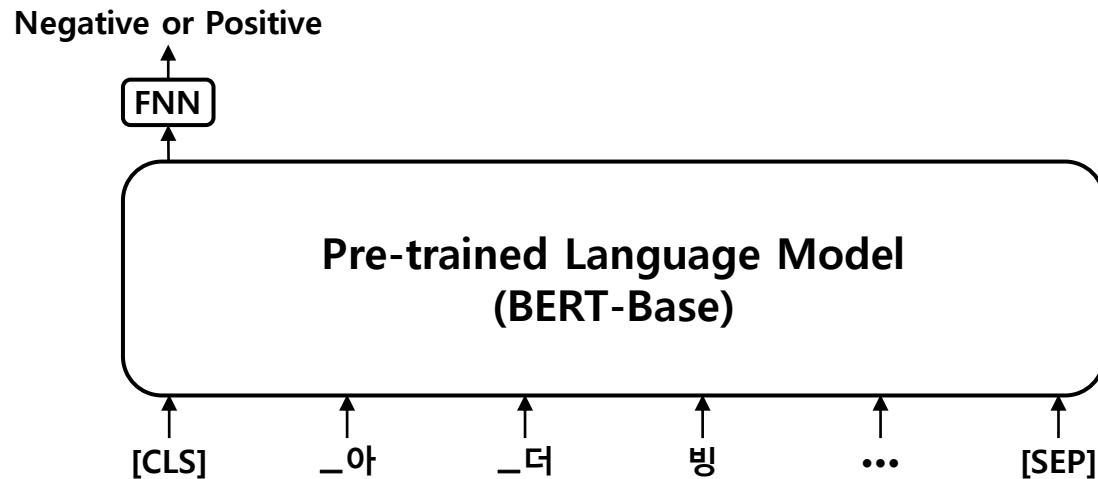


예제 코드 다운로드:
<https://github.com/KUNLP/KTAI-Practice>



실습: Sentiment Analysis

- 모델 구조



실습: Sentiment Analysis

모델 설계

```
1 import torch.nn as nn
2 from transformers import BertPreTrainedModel, BertModel
3
4
5 class SentimentClassifier(BertPreTrainedModel):
6
7     def __init__(self, config):
8         super(SentimentClassifier, self).__init__(config)
9
10        # BERT 모델
11        self.bert = BertModel(config)
12
13        # 히든 사이즈
14        self.hidden_size = config.hidden_size
15
16        # 분류할 라벨의 개수
17        self.num_labels = config.num_labels
18
19        self.linear = nn.Linear(in_features=self.hidden_size, out_features=self.num_labels)
20
21    def forward(self, input_ids):
22        outputs = self.bert(input_ids=input_ids)
23
24        # (batch_size, max_length, hidden_size)
25        bert_output = outputs[0]
26
27        # (batch_size, hidden_size)
28        cls_vector = bert_output[:, 0, :]
29
30        # class_output : (batch_size, num_labels)
31        cls_output = self.linear(cls_vector)
32
33        return cls_output
```

BERT 사전학습 모델 생성자
오버라이딩

• `input_ids` (`torch.LongTensor` of shape `(batch_size, sequence_length)`) -
Indices of input sequence tokens in the vocabulary.

`last_hidden_state` (`torch.FloatTensor` of shape `(batch_size, sequence_length, hidden_size)`) -
Sequence of hidden-states at the output of the last layer of the model.



실습: Sentiment Analysis


데이터 읽기

```
def read_data(file_path):  
    with open(file_path, "r", encoding="utf8") as inFile:  
        lines = inFile.readlines()  
  
    datas = []  
    for line in lines:  
        # 입력 데이터를 #을 기준으로 분리  
        pieces = line.strip().split("#")  
  
        # 리뷰, 정답  
        input_sequence, label = pieces[0].split(" "), pieces[1]  
  
        datas.append((input_sequence, label))  
  
    return datas
```

_아 _더 빙 .. _진짜 _짜 증 나 네요 _목소리 negative

사전 읽기

```
def read_vocab_data(vocab_data_path):  
    term2idx, idx2term = {}, {}  
  
    with open(vocab_data_path, "r", encoding="utf8") as inFile:  
        lines = inFile.readlines()  
  
    for line in lines:  
        term = line.strip()  
        term2idx[term] = len(term2idx)  
        idx2term[term2idx[term]] = term  
  
    return term2idx, idx2term
```



label_vocab.txt	
1	negative
2	positive
3	



실습: Sentiment Analysis

데이터 전처리

언어모델에 포함된 객체

```
def convert_data2feature(datas, max_length, tokenizer, label2idx):  
    input_ids_features, label_id_features = [], []
```

“아 “더 Bing .. “진짜 “짜 증 나 네요 “목소리

negative

```
for input_sequence, label in datas:
```

CLS, SEP 토큰 추가

```
tokens = [tokenizer.cls_token]
```

```
tokens += input_sequence
```

```
tokens = tokens[:max_length - 1]
```

```
tokens += [tokenizer.sep_token]
```

정해진 max length 보다
길면 삭제 (보통 512)

word piece들을 대응하는 index로 치환

```
input_ids = tokenizer.convert_tokens_to_ids(tokens)
```

padding 생성

```
padding = [tokenizer._convert_token_to_id(tokenizer.pad_token)] * (max_length - len(input_ids))
```

```
input_ids += padding
```

```
label_id = label2idx[label]
```

변환한 데이터를 각 리스트에 저장

```
input_ids_features.append(input_ids)
```

```
label_id_features.append(label_id)
```

tokenizing된 리뷰 데이터["아", "더", "Bing", "...", "진짜", "짜", "증", "나", "네요", "목소리", ...]
input_ids : [2, 3360, 28709, 18, 18, 12704, 29334, ...]
label_id : [1]



실습: Sentiment Analysis

Train

```
def train(config):  
    # BERT config 객체 생성  
    bert_config = BertConfig.from_pretrained(pretrained_model_name_or_path=config["pretrained_model_name_or_path"],  
                                             cache_dir=config["cache_dir_path"])  
    setattr(bert_config, "num_labels", config["num_labels"])  
    # BERT tokenizer 객체 생성  
    bert_tokenizer = KoBertTokenizer.from_pretrained(pretrained_model_name_or_path=config["pretrained_model_name_or_path"],  
                                                    cache_dir=config["cache_dir_path"])  
    # 라벨 딕셔너리 생성  
    label2idx, idx2label = read_vocab_data(vocab_data_path=config["label_vocab_data_path"])  
    # 학습 및 평가 데이터 읽기  
    train_datas = read_data(file_path=config["train_data_path"])  
    # 입력 데이터 전처리  
    train_input_ids_features, train_label_id_features = convert_data2feature(datas=train_datas,  
                                                                              max_length=config["max_length"],  
                                                                              tokenizer=bert_tokenizer,  
                                                                              label2idx=label2idx)
```

기존 BERT config에 프로그램에서 새로 정의한 config 추가



실습: Sentiment Analysis

Train

학습 데이터를 batch 단위로 추출하기 위한 DataLoader 객체 생성

```
train_dataset = TensorDataset(train_input_ids_features, train_label_id_features)
train_dataloader = DataLoader(dataset=train_dataset, batch_size=config["batch_size"],
                              sampler=RandomSampler(train_dataset))
```

사전 학습된 BERT 모델 파일로부터 가중치 불러옴

```
model = SentimentClassifier.from_pretrained(pretrained_model_name_or_path=config["pretrained_model_name_or_path"],
                                           cache_dir=config["cache_dir_path"], config=bert_config).cuda()
```

loss를 계산하기 위한 함수

```
loss_func = nn.CrossEntropyLoss()
```

모델 학습을 위한 optimizer

```
optimizer = optim.Adam(model.parameters(), lr=2e-5)
```

```
for epoch in range(config["epoch"]):
```

```
    model.train()
```

학습모드

```
    total_loss = []
```

```
    for batch in train_dataloader:
```

```
        batch = tuple(t.cuda() for t in batch)
```

```
        input_ids, label_id = batch
```

```
    # 역전파 단계를 실행하기 전에 변화도를 0으로 변경
    optimizer.zero_grad()
```

```
    # 모델 예측 결과
```

```
    hypothesis = model(input_ids)
```

```
    # loss 계산
```

```
    loss = loss_func(hypothesis, label_id)
```

```
    # loss 값으로부터 모델 내부 각 매개변수에 대하여 gradient 계산
    loss.backward()
```

```
    # 모델 내부 각 매개변수 가중치 갱신
```

```
    optimizer.step()
```



실습: Sentiment Analysis

Test

```
def test(config):
    # BERT config 객체 생성
    bert_config = BertConfig.from_pretrained(pretrained_model_name_or_path=config["save_dir_path"],
                                             cache_dir=config["cache_dir_path"])

    # BERT tokenizer 객체 생성
    bert_tokenizer = KoBertTokenizer.from_pretrained(pretrained_model_name_or_path=config["pretrained_model_name_or_path"],
                                                    model.eval())

    # 라벨 딕셔너리 생성
    label2idx, idx2label = read_vocab_

    # 평가 데이터 읽기
    test_datas = read_data(file_path=
    test_datas = test_datas[:100]

    # 입력 데이터 전처리
    test_input_ids_features, test_label

    # 학습한 모델 파일로부터 가중치 불러오기
    model = SentimentClassifier.from_pretrained(

    # 평가모드
    for batch in test_dataloader:
        batch = tuple(t.cuda() for t in batch)
        input_ids, label_id = batch

        with torch.no_grad():
            # 모델 예측 결과
            hypothesis = model(input_ids)
            # 모델의 출력값에 softmax와 argmax 함수를 적용
            hypothesis = torch.argmax(torch.softmax(hypothesis, dim=-1), dim=-1)

        # Tensor를 리스트로 변경
        hypothesis = hypothesis.cpu().detach().numpy().tolist()
        label_id = label_id.cpu().detach().numpy().tolist()

        for index in range(len(input_ids)):
            input_tokens = bert_tokenizer.convert_ids_to_tokens(input_ids[index])
            input_sequence = bert_tokenizer.convert_tokens_to_string(input_tokens[1:input_tokens.index(bert_tokenizer.sep_token)])
            predict = idx2label[hypothesis[index]]
            correct = idx2label[label_id[index]]

            print("입력 : {}".format(input_sequence))
            print("출력 : {}, 정답 : {}\n".format(predict, correct))
```

dim (int) – the dimension to reduce. If None, the argmax of the flattened input is returned.

맨 마지막 차원으로

```
>>> a = torch.randn(4, 4)
>>> a
tensor([[ 1.3398,  0.2663, -0.2686,  0.2450],
        [-0.7401, -0.8805, -0.3402, -1.1936],
        [ 0.4907, -1.3948, -1.0691, -0.3132],
        [-1.6092,  0.5419, -0.2993,  0.3195]])
>>> torch.argmax(a, dim=1)
tensor([ 0,  2,  0,  1])
```



실습: Sentiment Analysis

Main

```
if(__name__=="__main__"):
    output_dir = os.path.join(root_dir, "output")
    cache_dir = os.path.join(root_dir, "cache")

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    if not os.path.exists(cache_dir):
        os.makedirs(cache_dir)

    config = {"mode": "train",
              "train_data_path": os.path.join(root_dir, "train_datas_wordpiece.txt"),
              "test_data_path": os.path.join(root_dir, "test_datas_wordpiece.txt"),
              "output_dir_path": output_dir,
              "cache_dir_path": cache_dir,
              "pretrained_model_name_or_path": "monologg/kobert",
              "label_vocab_data_path": os.path.join(root_dir, "label_vocab.txt"),
              "num_labels": 2,
              "max_length": 142,
              "epoch": 10,
              "batch_size": 64
             }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

과제

1. 빈칸 채우기: 7점
2. 성능 개선: 10점 (ipynb에 개선 포인트 기술, 기존 대비 성능 개선 결과 제시)

입력 : 최고!!!!!!!!!!!!!!!!!!!!

출력 : positive, 정답 : positive

입력 : 발연기 도저히 못보겠다 진짜 이렇게 연기를 못할거라곤 상상도 못했네

출력 : negative, 정답 : negative

입력 : 나이스

출력 : negative, 정답 : positive

입력 : 별 재미도없는거 우려먹어 챔프에서 방송 몇번했더라 ? ㅋㅋㅋㅋ

출력 : negative, 정답 : negative

입력 : '13일의 금요일', '나이트메어' 시리즈와 함께 가장 많은 시리즈를 양산하

출력 : positive, 정답 : positive



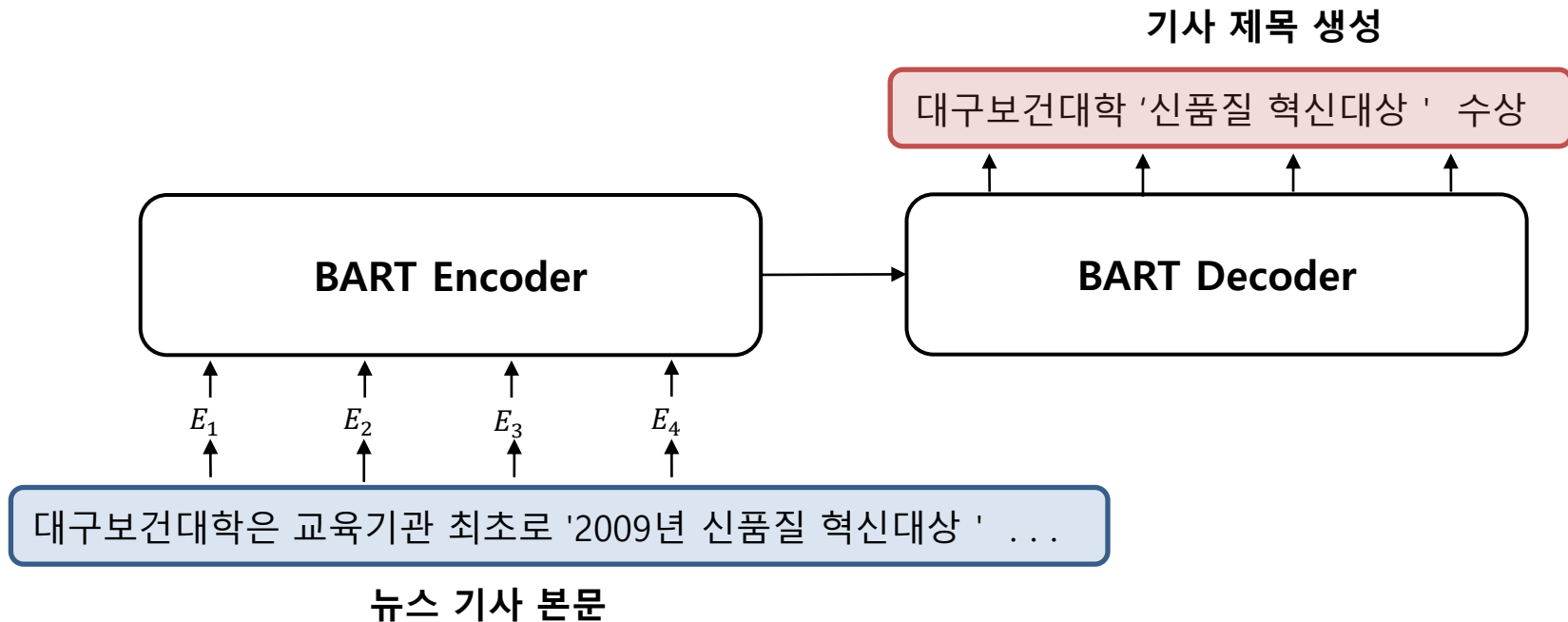
Pretrained Language Model: PART II

Practical Exercise

실습: Title Generation

- BART를 이용한 뉴스기사 제목 생성기

예제 코드 다운로드:
<https://github.com/KUNLP/KTAI-Practice>



실습: Title Generation

모델 호출

```
from transformers import BartForConditionalGeneration
from transformers import PreTrainedTokenizerFast
def train(config):
    tokenizer = PreTrainedTokenizerFast.from_pretrained(config["pretrained_model_name_or_path"])
    model = BartForConditionalGeneration.from_pretrained(config["pretrained_model_name_or_path"]).cuda()
```

데이터 읽기

```
def read_data(file_path):
    datas = []
    with open(file_path, "r", encoding="utf8") as infile:
        for line in infile:
            # 입력 데이터를 #을 기준으로 분리
            pieces = line.strip().split("#")
            article, title = pieces[0], pieces[1]
            datas.append((article, title))

    return datas
```

별도의 모델 설계 없이
라이브러리 호출을 통해
모델 객체 생성!

대구보건대학은 교육기관 최초로 '2009년 신품질 혁신대상' ... ₩ 대구보건대학 '신품질 혁신대상' 수상



실습: Title Generation

데이터 전처리

```
def convert_data2feature(datas, max_length, max_dec_length, tokenizer):
    input_ids_features, attention_mask_features, decoder_input_features, decoder_attention_mask_features, label_features = [], [], [], [], []

    for article, title in tqdm(datas, desc="convert_data2feature"):
        # tokenizer를 사용하여 입력 문장을 word piece 단위로 분리
        tokenized_article = tokenizer.tokenize(article)
        tokenized_title = tokenizer.tokenize(title)

        # word piece들을 대응하는 index로 치환
        input_ids = tokenizer.convert_tokens_to_ids(tokenized_article)
        # padding을 제외한 실제 데이터 정보를 반영해주기 위한 attention mask
        attention_mask = [1] * len(input_ids)
        # 각 문장을 구분하기 위한 정보를 반영해주기 위한 token type

        decoder_input = tokenizer.convert_tokens_to_ids(['<s>'] + tokenized_title)
        decoder_attention_mask = [1] * len(decoder_input)
        label = tokenizer.convert_tokens_to_ids(tokenized_title + ['</s>'])

        # padding 생성
        padding = [tokenizer.convert_tokens_to_ids(tokenizer.pad_token)] * (max_length - len(input_ids))
        input_ids += padding
        attention_mask += padding

        padding = [tokenizer.convert_tokens_to_ids(tokenizer.pad_token)] * (max_dec_length - len(decoder_input))
        decoder_input += padding
        decoder_attention_mask += padding
        label += padding
```

Enc: 대구보건대학은 교육기관 최초로 '2009년 신품질 혁신대상' ...
→ Token ID로 변환 + Attention Mask

Dec Input: <s> 대구보건대학 '신품질 혁신대상' 수상
→ Token ID로 변환 + Attention Mask

Dec Label: 대구보건대학 '신품질 혁신대상' 수상 </s>



실습: Title Generation

데이터 전처리

```
# 변환한 데이터를 각 리스트에 저장
input_ids_features.append(input_ids[:max_length])
attention_mask_features.append(attention_mask[:max_length])
decoder_input_features.append(decoder_input[:max_dec_length])
decoder_attention_mask_features.append(decoder_attention_mask[:max_length])
label_features.append(label[:max_dec_length])

# 변환한 데이터를 Tensor 객체에 담아 반환
input_ids_features = torch.tensor(input_ids_features, dtype=torch.long)
attention_mask_features = torch.tensor(attention_mask_features, dtype=torch.long)
decoder_input_features = torch.tensor(decoder_input_features, dtype=torch.long)
decoder_attention_mask_features = torch.tensor(decoder_attention_mask_features, dtype=torch.long)
label_features = torch.tensor(label_features, dtype=torch.long)

return input_ids_features, attention_mask_features, decoder_input_features, decoder_attention_mask_features, label_features
```



실습: Title Generation

학습

```
def train(config):
    tokenizer = PreTrainedTokenizerFast.from_pretrained(config["pretrained_model_name_or_path"])
    model = BartForConditionalGeneration.from_pretrained(config["pretrained_model_name_or_path"]).cuda()
    """ Train the model """
    # 학습 및 평가 데이터 읽기
    train_datas = read_data(config["train_data_path"])
    test_datas = read_data(config["test_data_path"])

    # 입력 데이터 전처리
    train_input_ids_features, train_attention_mask_features, train_decoder_input_features, train_decoder_attention_mask_features, train_label_features = #
    convert_data2feature(train_datas, config["max_length"], config["max_dec_
    test_input_ids_features, test_attention_mask_features, test_decoder_input_f
    convert_data2feature(test_datas, config["max_length"], config["max_dec_

    # 학습 데이터를 batch 단위로 추출하기 위한 DataLoader 객체 생성
    train_features = TensorDataset(train_input_ids_features, train_attention_ma
    train_dataloader = DataLoader(train_features, sampler=RandomSampler(train_f
    batch_size=config["batch_size"])
```

별도의 모델 설계 없이
라이브러리 호출을 통해
모델 객체 생성!

Returns

A `Seq2SeqLMOutput` (If `return_dict=True` is passed or when the configuration (`BartConfig`) and Inputs.

- **loss** (`torch.FloatTensor` of shape `(1,)`, optional, returned when training.
- **logits** (`torch.FloatTensor` of shape `(batch_size, seq_length, vocab_size)`, returned when inference.)

```
for epoch in range(config["epoch"]):
    for step, batch in enumerate(train_dataloader):
        # Skip past any already trained steps if resuming training
        model.train()
        batch = tuple(t.cuda() for t in batch)
        outputs = model(input_ids=batch[0],
                        attention_mask=batch[1],
                        decoder_input_ids=batch[2],
                        decoder_attention_mask=batch[3],
                        labels=batch[4],
                        return_dict=True)

        loss = outputs["loss"]

        loss.backward()
        if (global_step+1) % 50 == 0:
            print("{} Processed.. Total Loss : {}".format(global_step+1, loss.item()))

        tr_loss += loss.item()

    optimizer.step()
    model.zero_grad()
    global_step += 1
```

Batch

텐서에서 스칼라
값을 추출하는
방법



실습: Title Generation

평가

```
def evaluate(config, model, tokenizer, test_dataloader):  
    model.eval()  
    for batch in tqdm(test_dataloader):  
        batch = tuple(t.cuda() for t in batch)
```

생성을 위한 별도의 함수 제공

```
        dec_outputs = model.generate(input_ids = batch[0],  
                                     attention_mask=batch[1],  
                                     max_length=config["max_dec_length"],  
                                     eos_token_id=1,  
                                     do_sample=False,  
                                     bad_words_ids=[[5]]  
        )
```

Greedy search할 것인지
Sampling할 것인지 여부

```
        batch_size = batch[0].size()[0]
```

나쁜 말 생성 금지!

```
        dec_outputs = dec_outputs.tolist()  
        dec_labels = batch[4].tolist()
```

처음부터 Padding Index까지만
끊어오기

```
        for index in range(batch_size):
```

```
            pred = "".join(tokenizer.convert_ids_to_tokens(dec_outputs[index][1:])).replace("<g>", " ").replace("<pad>", "").replace("</s>", "").replace("_", " ")  
            ref = "".join(tokenizer.convert_ids_to_tokens(dec_labels[index][: -1])).replace("<g>", " ").replace("<pad>", "").replace("</s>", "").replace("_", " ")
```

```
            print("REFERENCE : {}#nDECODED : {}#n".format(ref, pred))
```



실습: Title Generation

Main

```
if (__name__ == "__main__"):
    save_dir = os.path.join(root_dir, "save")
    output_dir = os.path.join(root_dir, "output")
    cache_dir = os.path.join(root_dir, "cache")

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)
    if not os.path.exists(cache_dir):
        os.makedirs(cache_dir)

    set_seed(seed=1234)

    config = {"mode": "train",
              "train_data_path": os.path.join(root_dir, "train.txt"),
              "test_data_path": os.path.join(root_dir, "test.txt"),
              "output_dir_path": output_dir,
              "save_dir_path": save_dir,
              "cache_dir_path": cache_dir,
              "pretrained_model_name_or_path": "hyunwoongko/kobart",
              "max_length": 250,
              "max_dec_length": 60,
              "epoch": 30,
              "batch_size": 16,
              }

    if (config["mode"] == "train"):
        train(config)
```

REFERENCE : 경제부/5. 11/2010 대한민국영어교육박람회 14일 개막..
DECODED : 2010 대한민국영어교육박람회 14일부터 3일간 엑스코서 개최

REFERENCE : “국정화 반대 시국선언 교사 징계”
DECODED : 교육부, 시국선언 교사들 파악...중징계 요구

REFERENCE : 미세먼지 걱정없는 필리핀의 고급 주거단지
DECODED : 포스코건설, 필리핀 클락 클락힐즈

REFERENCE : 한반도 상공 신호정보, 2020년까지 분석 시도
DECODED : 신호정보 수집 분석, 2020년까지 구축

REFERENCE : 로알앤컴퍼니, 장애인용 안전손잡이
DECODED : 로알앤컴퍼니 / 장애인용 안전손잡이 8종 출시

과제

빈칸 채우기: 10점



질의응답

Q&A

Homepage: <http://nlp.konkuk.ac.kr>
E-mail: nlpdrkim@konkuk.ac.kr

