data.py

```
from konlpy.tag import Okt
import pandas as pd
import tensorflow as tf
import enum
import os
import re
from sklearn.model_selection import train_test_split
import numpy as np
from configs import DEFINES

from tqdm import tqdm
FILTERS = "([~.,!?\"':;)(])"
PAD = "<PADDING>"
STD = "<START>"
END = "<END>"
UNK = "<UNKNOWN>"

PAD_INDEX = 0
STD_INDEX = 1
END_INDEX = 2
UNK_INDEX = 3

MARKER = [PAD, STD, END, UNK]
CHANGE_FILTER = re.compile(FILTERS)

# 판다스를 통해서 데이터를 불러와 학습 셋과 평가 셋으로
# 나누어 그 값을 리턴한다.
def load_data():
    data_df = pd.read_csv(DEFINES.data_path, header=0)
    question, answer = list(data_df['Q']), list(data_df['A'])
    train_input, eval_input, train_label, eval_label = \
        train_test_split(question, answer, test_size=0.33, random_state=42)
    return train_input, train_label, eval_input, eval_label

# Okt.morphs 함수를 통해 토크나이즈 된
# 리스트 객체를 받아 문자열을 재구성해서 리턴한다.
def prepro_like_morphlized(data):
    morph_analyzer = Okt()
    result_data = list()
    for seq in tqdm(data):
        morphlized_seq = " ".join(morph_analyzer.morphs(seq.replace(' ', '')))
        result_data.append(morphlized_seq)

    return result_data

# 인코딩 데이터를 만드는 함수이며
# 인덱스화 할 value 와 키가 단어이고 값이 인덱스인 딕셔너리를 받아
# 넘파이 배열에 인덱스화된 배열과 그 길이를 넘겨준다.
def enc_processing(value, dictionary):
    sequences_input_index = []
    sequences_length = []
```

```
    # 형태소 토크나이징 사용 유무
    if DEFINES.tokenize_as_morph:
        value = prepro_like_morphlized(value)

    for sequence in value:
        sequence = re.sub(CHANGE_FILTER, "", sequence)
        sequence_index = []

        # 문장을 스페이스 단위로 자르고 있다.
        for word in sequence.split():
            # 잘려진 단어들이 딕셔너리에 존재 하는지 보고
            # 그 값을 가져와 sequence_index 에 추가한다.
            if dictionary.get(word) is not None:
                sequence_index.extend([dictionary[word]])

            # 잘려진 단어가 딕셔너리에 존재 하지 않는
            # 경우 이므로 UNK(2)를 넣어 준다.
            else:
                sequence_index.extend([dictionary[UNK]])

        # 문장 제한 길이보다 길어질 경우 뒤에 토큰을 자르고 있다.
        if len(sequence_index) > DEFINES.max_sequence_length:
            sequence_index = sequence_index[:DEFINES.max_sequence_length]

        sequences_length.append(len(sequence_index))

        # max_sequence_length 보다 문장 길이가 작다면 빈 부분에 PAD(0)를 넣어준다.
        sequence_index += (DEFINES.max_sequence_length - len(sequence_index)) *
[dictionary[PAD]]
        sequences_input_index.append(sequence_index)

    # 인덱스화된 일반 배열을 넘파이 배열로 변경한다.
    # 이유는 텐서플로우 dataset 에 넣어 주기 위한 사전 작업이다.
    return np.asarray(sequences_input_index), sequences_length


# 디코딩 입력 데이터를 만드는 함수이다.
def dec_input_processing(value, dictionary):
    sequences_output_index = []
    sequences_length = []

    if DEFINES.tokenize_as_morph:
        value = prepro_like_morphlized(value)

    for sequence in value:
        sequence = re.sub(CHANGE_FILTER, "", sequence)
        sequence_index = []

        # 디코딩 입력의 처음에는 START 가 와야 하므로
        # 그 값을 넣어 주고 시작한다.
        sequence_index = [dictionary[STD]] + [dictionary[word] for word in
sequence.split()]
```

```
        if len(sequence_index) > DEFINES.max_sequence_length:
            sequence_index = sequence_index[:DEFINES.max_sequence_length]
        sequences_length.append(len(sequence_index))
        sequence_index += (DEFINES.max_sequence_length - len(sequence_index)) *
[dictionary[PAD]]
        sequences_output_index.append(sequence_index)

    return np.asarray(sequences_output_index), sequences_length

# 디코딩 출력 데이터를 만드는 함수이다.
def dec_target_processing(value, dictionary):
    sequences_target_index = []

    if DEFINES.tokenize_as_morph:
        value = prepro_like_morphlized(value)
    for sequence in value:
        sequence = re.sub(CHANGE_FILTER, "", sequence)

        # 문장에서 스페이스 단위별로 단어를 가져와서
        # 딕셔너리의 값인 인덱스를 넣어 준다.
        # 디코딩 출력의 마지막에 END 를 넣어 준다.
        sequence_index = [dictionary[word] for word in sequence.split()]

        # 문장 제한 길이보다 길어질 경우 뒤에 토큰을 자르고 있다.
        # 그리고 END 토큰을 넣어 준다
        if len(sequence_index) >= DEFINES.max_sequence_length:
            sequence_index = sequence_index[:DEFINES.max_sequence_length-1] +
[dictionary[END]]
        else:
            sequence_index += [dictionary[END]]

        # max_sequence_length 보다 문장 길이가
        sequence_index += (DEFINES.max_sequence_length - len(sequence_index)) *
[dictionary[PAD]]
        sequences_target_index.append(sequence_index)

    return np.asarray(sequences_target_index)


# 인덱스를 스트링으로 변경하는 함수이다.
def pred2string(value, dictionary):
    sentence_string = []
    for v in value:
        # 딕셔너리에 있는 단어로 변경해서 배열에 담는다.
        sentence_string = [dictionary[index] for index in v['indexs']]

    print(sentence_string)
    answer = ""

    # 패딩값과 엔드값이 담겨 있으므로 패딩은 모두 스페이스 처리 한다.
    for word in sentence_string:
        if word not in PAD and word not in END:
```

```
            answer += word
            answer += " "

    print(answer)
    return answer

# 데이터 각 요소에 대해서 rearrange 함수를
# 통해서 요소를 변환하여 맵으로 구성한다.
def rearrange(input, output, target):
    features = {"input": input, "output": output}
    return features, target

# 학습에 들어가 배치 데이터를 만드는 함수이다.
def train_input_fn(train_input_enc, train_output_dec, train_target_dec, batch_size):
    # Dataset 을 생성하는 부분으로써 from_tensor_slices 부분은
    # 각각 한 문장으로 자른다고 보면 된다.
    # train_input_enc, train_output_dec, train_target_dec
    # 3 개를 각각 한문장으로 나눈다.
    dataset = tf.data.Dataset.from_tensor_slices((train_input_enc, train_output_dec,
train_target_dec))
    dataset = dataset.shuffle(buffer_size=len(train_input_enc))

    # 배치 인자 값이 없다면  에러를 발생 시킨다.
    assert batch_size is not None, "train batchSize must not be None"

    # from_tensor_slices 를 통해 나눈것을 배치크기 만큼 묶어 준다.
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(rearrange)

    # repeat()함수에 원하는 에포크 수를 넣을수 있으면
    # 아무 인자도 없다면 무한으로 이터레이터 된다.
    dataset = dataset.repeat()
    iterator = dataset.make_one_shot_iterator()

    # 이터레이터를 통해 다음 항목의 텐서 개체를 넘겨준다.
    return iterator.get_next()


# 평가에 들어가 배치 데이터를 만드는 함수이다.
def eval_input_fn(eval_input_enc, eval_output_dec, eval_target_dec, batch_size):
    dataset = tf.data.Dataset.from_tensor_slices((eval_input_enc, eval_output_dec,
eval_target_dec))

    # 전체 데이터를 섞는다.
    dataset = dataset.shuffle(buffer_size=len(eval_input_enc))
    assert batch_size is not None, "eval batchSize must not be None"
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(rearrange)

    # 평가이므로 1 회만 동작 시킨다.
    dataset = dataset.repeat(1)
```

```
    iterator = dataset.make_one_shot_iterator()
    return iterator.get_next()

# 토크나이징 해서 담을 배열을 생성하고
# 토그나이징과 정규표현식을 통해 만들어진 값들을 넘겨 준다.
def data_tokenizer(data):
    words = []
    for sentence in data:
        sentence = re.sub(CHANGE_FILTER, "", sentence)
        for word in sentence.split():
            words.append(word)
    return [word for word in words if word]

# 최초 사전 파일을 만드는 함수이며 파일이 존재 한다면 불러오는 함수이다.
def load_vocabulary():
    vocabulary_list = []
    # 사전 파일의 존재 유무를 확인한다.
    if (not (os.path.exists(DEFINES.vocabulary_path))):
        if (os.path.exists(DEFINES.data_path)):
            data_df = pd.read_csv(DEFINES.data_path, encoding='utf-8')
            question, answer = list(data_df['Q']), list(data_df['A'])

            # 질문과 응답 문장의 단어를 형태소로 바꾼다
            if DEFINES.tokenize_as_morph:
                question = prepro_like_morphlized(question)
                answer = prepro_like_morphlized(answer)

            data = []
            data.extend(question)
            data.extend(answer)
            words = data_tokenizer(data)
            words = list(set(words))

            # 데이터 없는 내용중에 MARKER 를 사전에
            # 추가 하기 위해서 아래와 같이 처리 한다.
            # 아래는 MARKER 값이며 리스트의 첫번째 부터
            # 순서대로 넣기 위해서 인덱스 0 에 추가한다.
            # PAD = "<PADDING>"
            # STD = "<START>"
            # END = "<END>"
            # UNK = "<UNKNOWN>"
            words[:0] = MARKER

        # 사전 리스트를 사전 파일로 만들어 넣는다.
        with open(DEFINES.vocabulary_path, 'w', encoding='utf-8') as vocabulary_file:
            for word in words:
                vocabulary_file.write(word + '\n')

    # 사전 파일이 존재하면 여기에서 그 파일을 불러서 배열에 넣어 준다.
    with open(DEFINES.vocabulary_path, 'r', encoding='utf-8') as vocabulary_file:
        for line in vocabulary_file:
```

```
            vocabulary_list.append(line.strip()) # strip()  양쪽 끝에 있는 공백과 \n
기호 삭제

    word2idx, idx2word = make_vocabulary(vocabulary_list)

    # 두가지 형태의 키와 값이 있는 형태를 리턴한다.
    # (예) 단어: 인덱스 , 인덱스: 단어)
    return word2idx, idx2word, len(word2idx)

# 리스트를 키가 단어이고 값이 인덱스인 딕셔너리를 만든다.
# 리스트를 키가 인덱스이고 값이 단어인 딕셔너리를 만든다.
def make_vocabulary(vocabulary_list):
    word2idx = {word: idx for idx, word in enumerate(vocabulary_list)}
    idx2word = {idx: word for idx, word in enumerate(vocabulary_list)}
    return word2idx, idx2word


def main(self):
    char2idx, idx2char, vocabulary_length = load_vocabulary()

if __name__ == '__main__':
    tf.logging.set_verbosity(tf.logging.INFO)
    tf.app.run(main)
```