Model.py

```
#-*- coding: utf-8 -*-
import tensorflow as tf
import sys
from configs import DEFINES

# 엘에스티엠(LSTM) 단층 네트워크 구성하는 부분
def make_lstm_cell(mode, hiddenSize, index):
    cell = tf.nn.rnn_cell.BasicLSTMCell(hiddenSize, name = "lstm"+str(index))
    if mode == tf.estimator.ModeKeys.TRAIN:
        cell = tf.contrib.rnn.DropoutWrapper(cell, output_keep_prob=DEFINES.dropout_width)
    return cell

# 에스티메이터 모델 부분이다.
def model(features, labels, mode, params):
    TRAIN = mode == tf.estimator.ModeKeys.TRAIN
    EVAL = mode == tf.estimator.ModeKeys.EVAL
    PREDICT = mode == tf.estimator.ModeKeys.PREDICT

    # 인코딩 부분 (미리 정의된 임베딩 벡터 사용 유무)
    if params['embedding'] == True:
        # 가중치 행렬에 대한 초기화 함수이다.
        # xavier (Xavier Glorot 와 Yoshua Bengio (2010)
        # URL : http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
        initializer = tf.contrib.layers.xavier_initializer()
        embedding = tf.get_variable(name = "embedding", # 이름
                                    shape=[params['vocabulary_length'], params['embedding_size']], #  모양
                                    dtype=tf.float32, # 타입
                                    initializer=initializer, # 초기화 값
```

```
                                    trainable=True) # 학습 유무
    else:
        # tf.eye 를 통해서 사전의 크기 만큼의 단위행렬
        # 구조를 만든다.
        embedding = tf.eye(num_rows = params['vocabulary_length'], dtype = tf.float32)
        embedding = tf.get_variable(name = "embedding", # 이름
                                    initializer = embedding, # 초기화 값
                                    trainable = False) # 학습 유무

    # 임베딩된 인코딩 배치를 만든다.
    embedding_encoder = tf.nn.embedding_lookup(params = embedding, ids = features['input'])

    # 임베딩된 디코딩 배치를 만든다.
    embedding_decoder = tf.nn.embedding_lookup(params = embedding, ids = features['output'])

    with tf.variable_scope('encoder_scope', reuse=tf.AUTO_REUSE):
        # 값이 True 이면 멀티레이어로 모델을 구성하고 False 이면
        # 단일레이어로 모델을 구성 한다.
        if params['multilayer'] == True:
            encoder_cell_list = [make_lstm_cell(mode, params['hidden_size'], i) for i in range(params['layer_size'])]
            rnn_cell = tf.contrib.rnn.MultiRNNCell(encoder_cell_list)
        else:
            rnn_cell = make_lstm_cell(mode, params['hidden_size'], "")

        # rnn_cell 에 의해 지정된 dynamic_rnn 반복적인 신경망을 만든다.
        # encoder_states 최종 상태  [batch_size, cell.state_size]
        encoder_outputs, encoder_states = tf.nn.dynamic_rnn(cell=rnn_cell, # RNN 셀
inputs=embedding_encoder, # 입력 값
```

```
                                                    dtype=tf.float32) # 타입


    with tf.variable_scope('decoder_scope', reuse=tf.AUTO_REUSE):
        if params['multilayer'] == True:
            decoder_cell_list = [make_lstm_cell(mode, params['hidden_size'], i) for
i in range(params['layer_size'])]
            rnn_cell = tf.contrib.rnn.MultiRNNCell(decoder_cell_list)
        else:
            rnn_cell = make_lstm_cell(mode, params['hidden_size'], "")


        decoder_initial_state = encoder_states
        decoder_outputs, decoder_states = tf.nn.dynamic_rnn(cell=rnn_cell, # RNN 셀
                    inputs=embedding_decoder, # 입력 값
                    initial_state=decoder_initial_state, # 인코딩의 마지막 값으로
초기화
                    dtype=tf.float32) # 타입



    # logits 는 마지막 히든레이어를 통과한 결과값이다.
    logits    =    tf.layers.dense(decoder_outputs,    params['vocabulary_length'],
activation=None)


        # argmax 를 통해서 최대 값을 가져 온다.
    predict = tf.argmax(logits, 2)


    if PREDICT:
        predictions = { # 예측 값들이 여기에 딕셔너리 형태로 담긴다.
            'indexs': predict, # 시퀀스 마다 예측한 값
        }
        return tf.estimator.EstimatorSpec(mode, predictions=predictions)


    #
    # logits 과 같은 차원을 만들어 마지막 결과 값과 정답 값을 비교하여 에러를 구한다.
```

```
    labels_ = tf.one_hot(labels, params['vocabulary_length'])
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,
labels=labels_))

    # 라벨과 결과가 일치하는지 빈도 계산을 통해 정확도를 측정하는 방법이다.
    accuracy = tf.metrics.accuracy(labels=labels, predictions=predict,name='accOp')

    # accuracy 를 전체 값으로 나눠 확률 값으로 한다.
    metrics = {'accuracy': accuracy}
    tf.summary.scalar('accuracy', accuracy[1])

    if EVAL:
        # 에러 값(loss)과 정확도 값(eval_metric_ops) 전달
        return tf.estimator.EstimatorSpec(mode, loss=loss, eval_metric_ops=metrics)

    # 수행 mode(tf.estimator.ModeKeys.TRAIN)가
    # 아닌 경우는 여기 까지 오면 안되도록 방어적 코드를 넣은것이다.
    assert TRAIN

    optimizer = tf.train.AdamOptimizer(learning_rate=DEFINES.learning_rate)
    train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())

    # 에러 값(loss)과 그라디언트 반환값 (train_op) 전달
    return tf.estimator.EstimatorSpec(mode, loss=loss, train_op=train_op)
```