Parameters - parameters.ini

[mode]
pretrained_word_embedding = True #사전 학습된 워드 임베딩의 사용 여부
word_embedding_path = ../../word_embedding/word_embedding.txt #워드 임베딩 저장 경로

[dataset]
dataset_text_folder = ../../data/Bigram_Data #데이터 저장 경로 (CONLL Format)

[ann]
char_embedding_dim = 25 #음절 단위 임베딩 차원
word_embedding_dim = 50 #바이그램 음절 단위 임베딩 차원

filter_size = 2 #음절 단위 자질 추출 CNN의 filter size(window size)
num_filters = 30#음절 단위 CNN모델의 feature map 개수

word_filter_size = 2,3 #음절 바이그램 단위 CNN 모델의 filter size
word_num_filters = 100 #음절 바이그램 단위 CNN 모델의 feature map 개수
l2_reg_lambda = 0.001 #l2_regularizer 바이어스 정하기

lstm_hidden_state_dim = 100 #LSTM(단방향) hidden state dimension

[training]
patience = 10 #10번 동안 early stop 참음
maximum_number_of_epochs = 100 #최대 100번 epoch
evaluation_every = 1000 #1000 step 마다 validation set과 비교
checkpoint_every = 1000 #1000 step 마다 학습 파라미터 저장
batch_size = 32 #mini batch size

dropout_rate = 0.5 #Dropout rate. Test, Validate 때는 무조건 1.0으로 설정

number_of_cpu_threads = 8 #CPU 사용 개수. 없어도 될듯

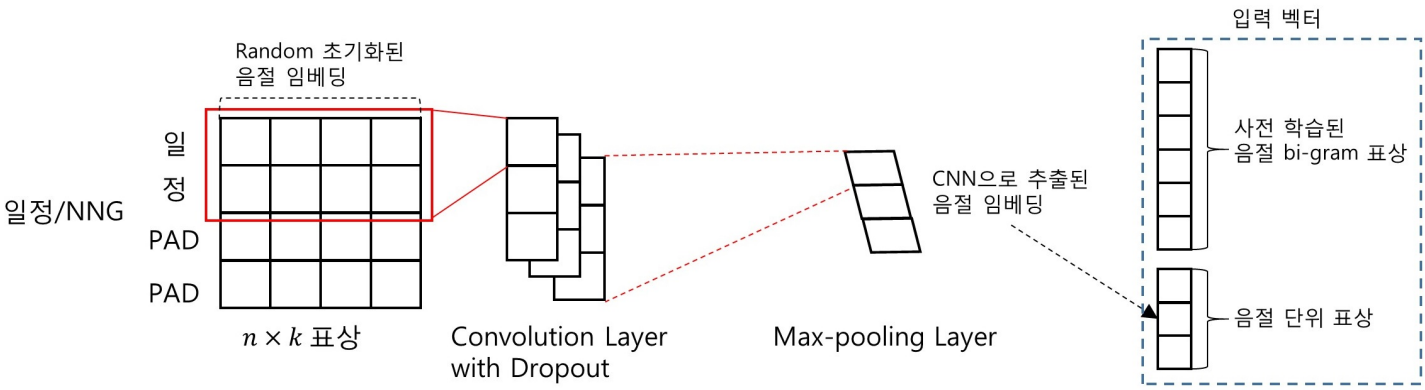
Model - cnn_lstm_crf.py

모델에는 6가지 placeholder(데이터 담는 그릇)가 필요.
1) 음절 바이그램 단위 입력 = input_word_x
2) 음절 단위 입력 = input_char_x
3) 실제 입력 길이 = input_word_lengths
4) speech act label = input_intent_y
5) slot bio tag = input_slot_y
6) dropout 확률 = dropout_keep_prob

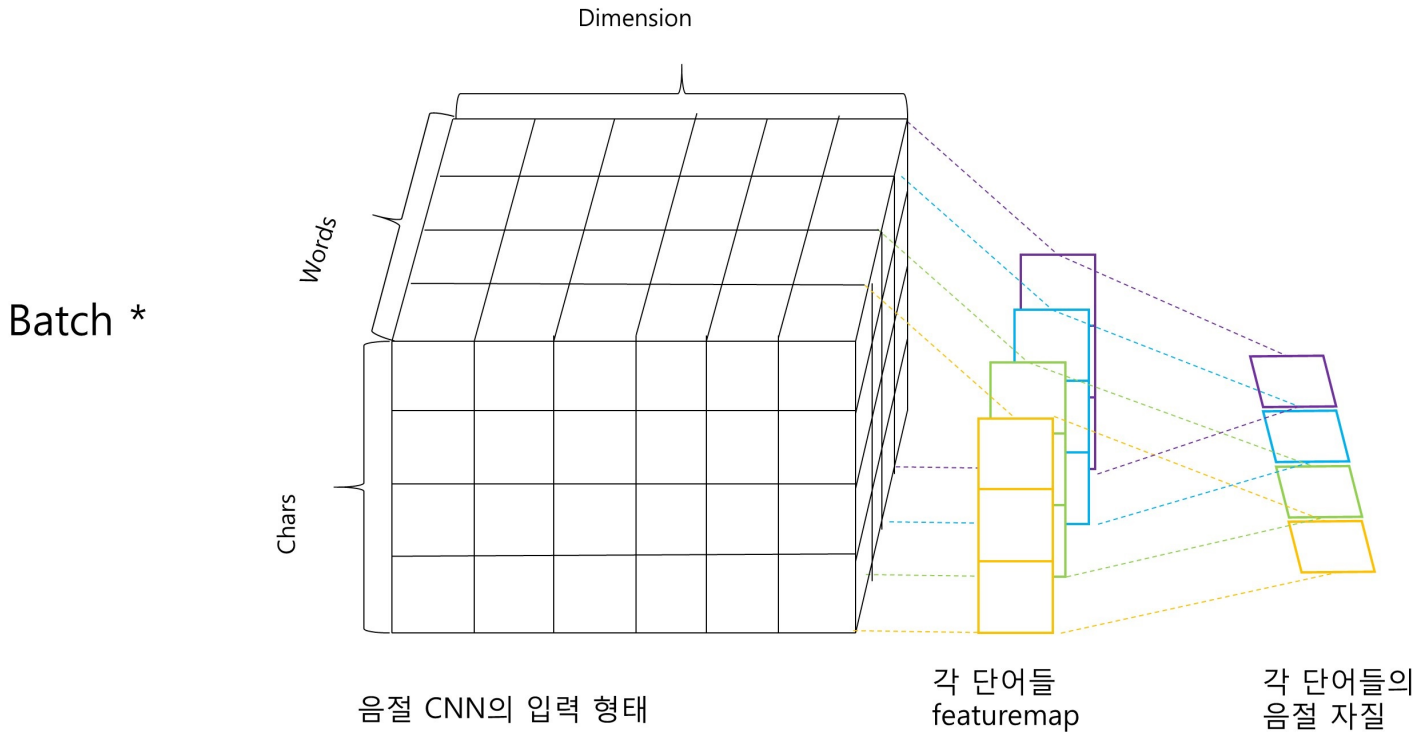
```
class Text_CNN_LSTM_CRF(object):
    def __init__(self, dataset, parameters):
        initializer = tf.contrib.layers.xavier_initializer()
        self.input_word_x = tf.placeholder(tf.int32, shape=[None,dataset.max_len],
name="input_word_x")#[?,]
        self.input_char_x = tf.placeholder(tf.int32, shape=
```

```
[None,dataset.max_len,dataset.max_word_len], name="input_char_x")
6       self.input_word_lengths = tf.placeholder(tf.int32, shape = [None],
name="input_word_lengths")
7       self.input_intent_y = tf.placeholder(tf.float32, shape=
[None,1,dataset.intent_number_of_classes], name="input_intent_y")#[15,]
8       self.input_slot_y = tf.placeholder(tf.float32, shape=
[None,dataset.max_len,dataset.slot_number_of_classes], name="input_slot_y")#[15,]
9       self.dropout_keep_prob = tf.placeholder(tf.float32,
name="dropout_keep_prob")#constant
```

입력 벡터 구성



음절 자질 추출 Convolutional Neural Network(CNN)



- 음절 임베딩을 가져온 후 음절 자질 추출(char_CNN)
- char_embedding_layer : 음절 임베딩을 가져옴
- embedded_char_expanded : embedded_char에 마지막 1차원 추가해준 이유 => tensorflow convolution input shape 맞춰주기 위함
- char_feature : 추출된 음절 자질

```
        with tf.variable_scope("char_embedding_layer"):
            self.char_embedding_weights = tf.get_variable("char_embedding_weights",
                    shape = [dataset.char_size, parameters['char_embedding_dim']],
                    initializer = initializer)
            embedded_char = tf.nn.embedding_lookup(self.char_embedding_weights,
self.input_char_x, name="embedded_jasos")#shape = [Batch, max sentence length, max word
length, char embedding dim]
            embedded_char_expanded = tf.expand_dims(embedded_char, -1)

        with tf.variable_scope("char_convolution_layer"):
            char_feature = char_CNN(
                embedded_char_expanded,
                parameters["char_embedding_dim"],
                parameters["filter_size"],
                parameters["num_filters"],
                initializer,
                dataset.max_word_len,
                dataset.max_len,
                self.dropout_keep_prob)
```

음절 자질과 사전 학습된 담어 임베딩 연결하기

- 추출한 음절 자질과 사전 학습된 음절 바이그램 임베딩 연결
- 1 ~ 6 : 문장의 입력 음절 bi-gram의 임베딩을 가져옴 = embedded_tokens
- 8 ~ 9 : Bidirectional LSTM의 입력으로 들어갈 음절 임베딩과 음절 bi-gram의 입력 임베딩을 연결(concat)

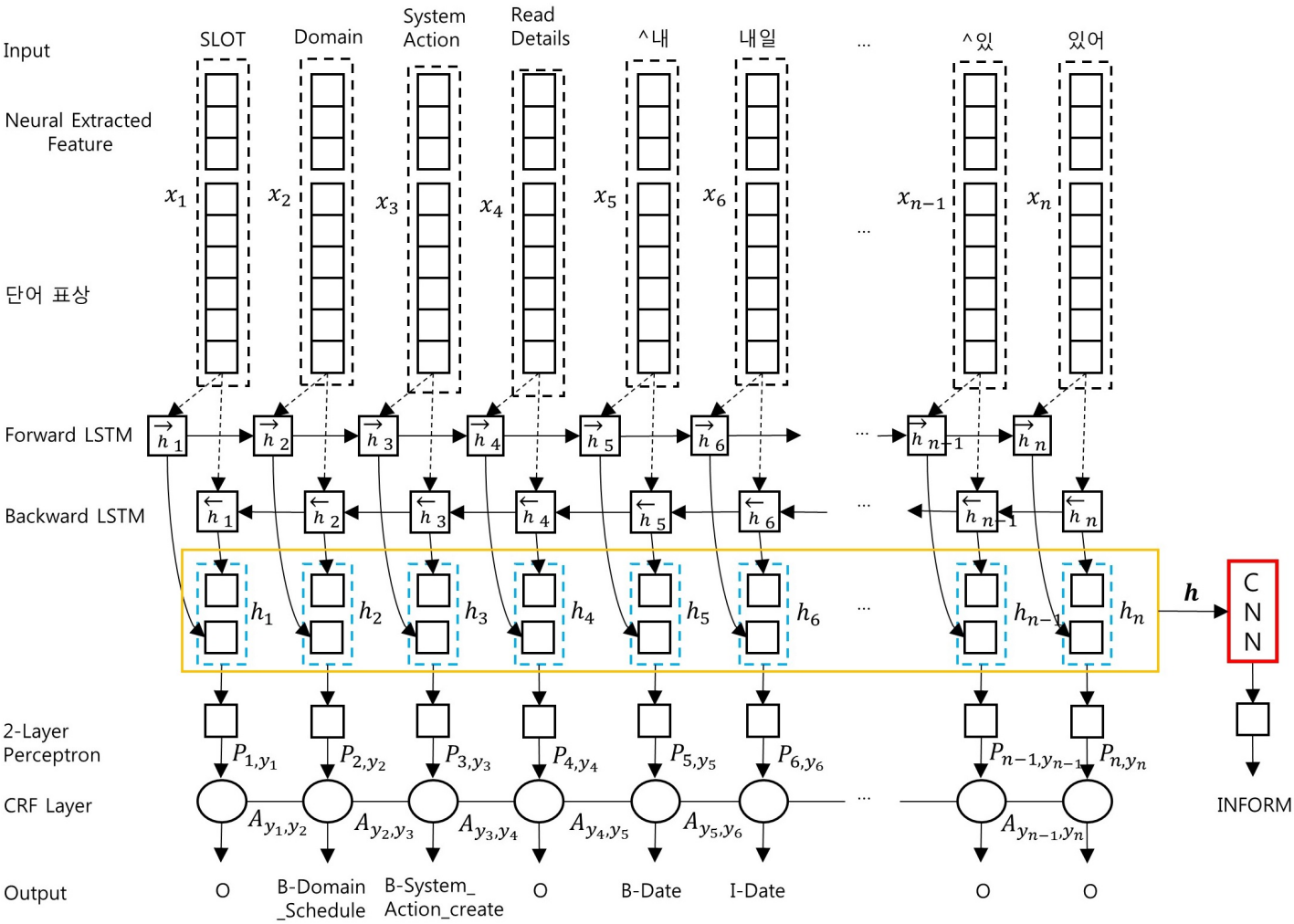
```
        with tf.variable_scope("word_embedding_layer"):
            self.embedding_weights = tf.get_variable("embedding_weights",
                    shape = [dataset.vocabulary_size,parameters["word_embedding_dim"]],
                    initializer = initializer)

            embedded_tokens = tf.nn.embedding_lookup(self.embedding_weights,
self.input_word_x, name="embedded_tokens")

        with tf.variable_scope("concatentate_char_word_embedding"):
            token_lstm_input = tf.concat([char_feature,embedded_tokens], axis=2,
name="token_fc_input")

        with tf.variable_scope("token_dropout"):
            token_lstm_input_drop = tf.nn.dropout(token_lstm_input,
self.dropout_keep_prob, name="token_lstm_input_drop")
```

본 모델



LSTM을 통한 문장 표현 학습

- 입력 임베딩 bi_lstm의 입력으로 주어짐
  [입력값]
- 입력 임베딩 : token_lstm_input_drop
- parameters['lstm_hidden_state_dim'] : LSTM의 hidden state 차원
- self.input_word_lengths : 실제 입력 문장의 길이
- initializer : 초기화 방식
  [반환값]
- lstm_inputs : LSTM의 매 timestep의 hidden state들의 concat 행렬 ($h_1 \bigoplus \ldots \bigoplus h_n$)
- lstm_last_output : LSTM의 마지막 출력 벡터 h_n

```
1        with tf.variable_scope("token_lstm_layer"):
2            lstm_outputs,lstm_last_output = bi_LSTM(
3                    token_lstm_input_drop,
4                    parameters['lstm_hidden_state_dim'],
5                    self.input_word_lengths,
6                    initializer)
```

CNN을 통한 화행 분류

- 화행을 분류하기 위해 Bi-LSTM의 출력 행렬을 Convolutional Neural Network의 입력으로 주어준다.
- 2 : 화행 분류 CNN의 입력으로 주어지기 위해 입력 벡터 차원 확장
- 3 ~ 9 : 화행 분류 CNN 함수 호출

[입력값]

- lstm_cnn_input : Bi-LSTM의 출력 벡터들 H 행렬
- lstm_cnn_input.get_shape().as_list()[2] : 입력 벡터의 dimension
- parameters["word_filter_size"] : window size
- parameters["word_num_filters"] : window size 함수를 적용해 생성되는 feature map의 개수
- dataset.max_len : 학습 데이터의 최대 문장 길이

[반환값]

- cnn_feature : max pooling된 벡터값
- num_filters_total : cnn_feature 벡터 크기
- 14 ~ 18 : CNN의 출력으로 얻은 max pooling 벡터를 classification하기 위해 가중치 행렬과 bias
- 20 ~ 21 : 각 화행 label들의 점수들을 얻고, 그 중 제일 최대인 값을 예측값으로 추출

```
1        with tf.variable_scope("Intent_Convolution_Layer"):
2            lstm_cnn_input = tf.expand_dims(lstm_outputs,-1)
3            cnn_feature, num_filters_total = intent_CNN(
4                    lstm_cnn_input,
5                    lstm_cnn_input.get_shape().as_list()[2],
6                    parameters["word_filter_size"],
7                    parameters["word_num_filters"],
8                    initializer,
9                    dataset.max_len)
10
11       with tf.variable_scope("cnn_drop_out"):
12           h_drop = tf.nn.dropout(cnn_feature, self.dropout_keep_prob)
13
14       with tf.variable_scope("Intent_Classifier"):
15           W = tf.get_variable("W",
16                   shape = [num_filters_total, dataset.intent_number_of_classes],
17                   initializer = initializer)
18           b = tf.Variable(tf.constant(0.1, shape =
   [dataset.intent_number_of_classes]), name = "b")
19
20           self.intent_scores = tf.nn.xw_plus_b(h_drop, W, b, name="scores")
21           self.intent_predictions = tf.argmax(tf.nn.softmax(self.intent_scores),1,name
   = "predictions")
```

CRF를 통한 슬롯 채우기

- TensorFlow에서 라이브러리로 지원하는 CRF는 Linear-chain CRF이다.
- Linear-chain CRF 자세한 방법은 블로그를 참조 https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html
- 결론적으론 Observation score와 transition score가 필요하다고 생각하면 된다.
- Observation score : 매 time step마다 슬롯(BIO Tag) Label로 분류될 점수들, Fully connected NN을 이용하여 점수 추출한다. (1 ~ 16)
- Transition score : 이전 time step의 label에서 현재 time step의 label이 나올 전이 확률 점수
- Transition score는 처음에 랜덤값으로 초기화하여 학습을 통해 최적 가중치 값을 찾아간다.

23 : crf라이브러리를 이용하여 에러값과 학습된 transition score matrix를 반환 받는다.

- crf_input : 입력(Observation matrix)
- crf_tag_input : 정답

- crf_input_length : 길이

```
        with tf.variable_scope("Slot_Filling_Bi_LSTM_1"):
            lstm_output = tf.reshape(lstm_outputs, [-1,2*parameters['lstm_hidden_state_dim']]) #batch*leng , dim
            W = tf.get_variable("W",
                    shape = [2 * parameters['lstm_hidden_state_dim'], parameters['lstm_hidden_state_dim']],
                    initializer = initializer)
            b = tf.Variable(tf.constant(0.0 , shape = [parameters['lstm_hidden_state_dim']]), name= "b")
            before_slot_labeling = tf.nn.xw_plus_b(lstm_output, W,b,name="before_slot_tagging")

        with tf.variable_scope("Slot_Filling_Bi_LSTM_2"):
            W = tf.get_variable("W",
                    shape = [parameters['lstm_hidden_state_dim'], dataset.slot_number_of_classes],
                    initializer = initializer)
            b = tf.Variable(tf.constant(0.0 , shape = [dataset.slot_number_of_classes]), name= "b")

            self.slot_scores = tf.nn.xw_plus_b(before_slot_labeling, W, b, name="slot_scores")
            self.unary_scores = tf.reshape(self.slot_scores, [-1, dataset.max_len, dataset.slot_number_of_classes], name="unary_scores")

        with tf.variable_scope("CRF_Layer"):
            crf_input = tf.reshape(self.slot_scores, [-1, dataset.max_len, dataset.slot_number_of_classes], name="crf_input")
            crf_input_length = self.input_word_lengths
            crf_tag_input = tf.argmax(self.input_slot_y,2)
            crf_tag_input = tf.cast(crf_tag_input, tf.int32)
            log_likelihood, transition_params = tf.contrib.crf.crf_log_likelihood(
                crf_input, crf_tag_input, crf_input_length)
            self.slot_log_likelihood = log_likelihood
            self.slot_transition_params = transition_params
```

Loss 값 계산

- 화행 분류의 loss값은 cross entropy를 이용하였다.
- 슬롯 채우기의 loss값은 crf 라이브러리를 사용해 얻은 반환값 log_likelihood
- 최종 loss값을 두개의 합으로 계산하여 학습을 진행하도록 하였다.

```
        with tf.variable_scope("intent_loss"):
            self.intent_losses = tf.nn.softmax_cross_entropy_with_logits(logits = self.intent_scores, labels = self.input_intent_y)
            self.intent_loss = tf.reduce_mean(self.intent_losses)
        with tf.variable_scope("intent_accuracy"):
            self.input_intent_y_squeeze = tf.squeeze(self.input_intent_y,[1])
            correct_predictions = tf.equal(self.intent_predictions, tf.argmax(self.input_intent_y_squeeze,1))
            self.intent_accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")

        with tf.variable_scope("slot_loss"):
            self.slot_loss = tf.reduce_mean(-log_likelihood)
```

```
11
12      with tf.variable_scope("total_loss"):
13          self.loss =  self.intent_loss +  self.slot_loss
```