Cem Birler
1897119227

Homework #5 Report

My first attempt was to use the given simple RNN architecture with state size 100 and embedding size 50. However, this result gave me around 88% accuracy on Japanese and Italian. Later I have changed my model and tried LSTM. This improved my result to around 92% for both languages and then I have changed and used Bidirectional LSTM since language doesn’t only have forward dependencies but also backward.

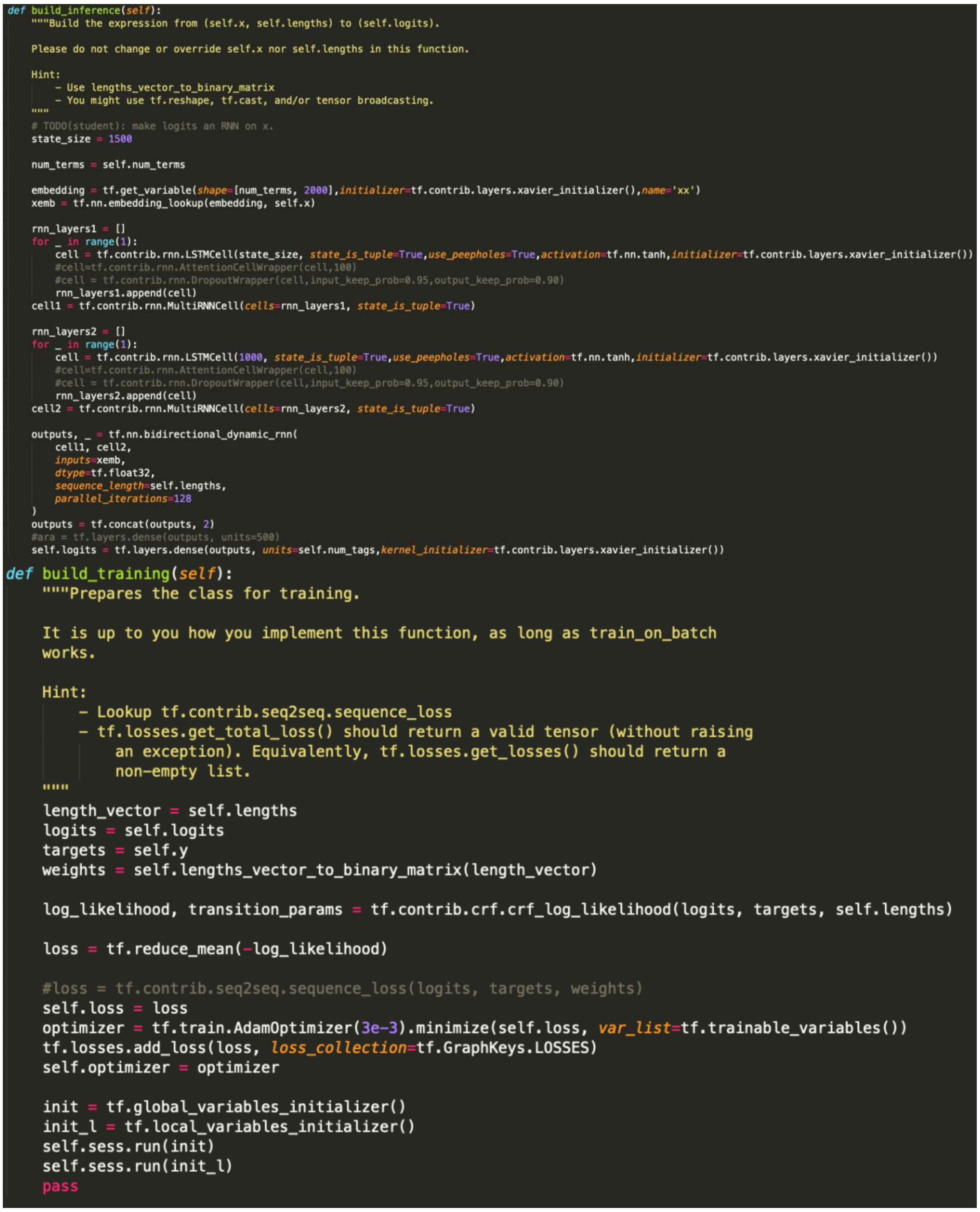
Before using Bidirectional LSTM, I have played with the state size number, batch size, learning rate and added another dense layer before the final output layer. But later on, I concluded that adding another dense layer didn’t affect my accuracy and dropped it in order to not lose speed. I also experimented with tf.contrib.rnn.AttentionCellWrapper and tf.contrib.rnn.DropoutWrappers after each LSTM layer, although I didn’t see any significant change in the accuracy if I had more time I would experiment more with different combinations of number of units and keep probability.

When I searched online for state-of-the-art POS taggers, I came across with the fully connected Conditional Random Field (CRF) layer instead of plain fully connected layer and used TF implementation to calculate the loss instead of seq2seq loss. This increased my accuracy around 0.5 even though I am not exactly sure why so. Also before every epoch I shuffle my dataset which slight increased my overall accuracy.

LATEST SUBMISSION MODEL DETAILS:

Word embedding size 2000
Bidirectional LSTM with each state size of 1500 and initialized by Xavier initialization
Bidirectional LSTM is run with 128 parallel iterations to train more.
Dense layer with Xavier initialization
Adam Optimizer with learning rate 0.003
Shuffle dataset before every epoch
Batch size 64
Loss is measured with CRF log likelihood.

Below you can find my exact implementation of build_inference and build_training functions.



```python
def build_inference(self):
    """Build the expression from (self.x, self.lengths) to (self.logits).

    Please do not change or override self.x nor self.lengths in this function.

    Hint:
        - Use lengths_vector_to_binary_matrix
        - You might use tf.reshape, tf.cast, and/or tensor broadcasting.
    """
    # TODO(student): make logits an RNN on x.
    state_size = 1500

    num_terms = self.num_terms

    embedding = tf.get_variable(shape=[num_terms, 2000],initializer=tf.contrib.layers.xavier_initializer(),name='xx')
    xemb = tf.nn.embedding_lookup(embedding, self.x)

    rnn_layers1 = []
    for _ in range(1):
        cell = tf.contrib.rnn.LSTMCell(state_size, state_is_tuple=True,use_peepholes=True,activation=tf.nn.tanh,initializer=tf.contrib.layers.xavier_initializer())
        #cell=tf.contrib.rnn.AttentionCellWrapper(cell,100)
        #cell = tf.contrib.rnn.DropoutWrapper(cell,input_keep_prob=0.95,output_keep_prob=0.90)
        rnn_layers1.append(cell)
    cell1 = tf.contrib.rnn.MultiRNNCell(cells=rnn_layers1, state_is_tuple=True)

    rnn_layers2 = []
    for _ in range(1):
        cell = tf.contrib.rnn.LSTMCell(1000, state_is_tuple=True,use_peepholes=True,activation=tf.nn.tanh,initializer=tf.contrib.layers.xavier_initializer())
        #cell=tf.contrib.rnn.AttentionCellWrapper(cell,100)
        #cell = tf.contrib.rnn.DropoutWrapper(cell,input_keep_prob=0.95,output_keep_prob=0.90)
        rnn_layers2.append(cell)
    cell2 = tf.contrib.rnn.MultiRNNCell(cells=rnn_layers2, state_is_tuple=True)

    outputs, _ = tf.nn.bidirectional_dynamic_rnn(
        cell1, cell2,
        inputs=xemb,
        dtype=tf.float32,
        sequence_length=self.lengths,
        parallel_iterations=128
    )
    outputs = tf.concat(outputs, 2)
    #ara = tf.layers.dense(outputs, units=500)
    self.logits = tf.layers.dense(outputs, units=self.num_tags,kernel_initializer=tf.contrib.layers.xavier_initializer())
def build_training(self):
    """Prepares the class for training.

    It is up to you how you implement this function, as long as train_on_batch
    works.

    Hint:
        - Lookup tf.contrib.seq2seq.sequence_loss
        - tf.losses.get_total_loss() should return a valid tensor (without raising
          an exception). Equivalently, tf.losses.get_losses() should return a
          non-empty list.
    """
    length_vector = self.lengths
    logits = self.logits
    targets = self.y
    weights = self.lengths_vector_to_binary_matrix(length_vector)

    log_likelihood, transition_params = tf.contrib.crf.crf_log_likelihood(logits, targets, self.lengths)

    loss = tf.reduce_mean(-log_likelihood)

    #loss = tf.contrib.seq2seq.sequence_loss(logits, targets, weights)
    self.loss = loss
    optimizer = tf.train.AdamOptimizer(3e-3).minimize(self.loss, var_list=tf.trainable_variables())
    tf.losses.add_loss(loss, loss_collection=tf.GraphKeys.LOSSES)
    self.optimizer = optimizer

    init = tf.global_variables_initializer()
    init_l = tf.local_variables_initializer()
    self.sess.run(init)
    self.sess.run(init_l)
    pass
```