Natural Language Processing
Final Deep Learning Assignment
Sequence to Sequence RNN for POS tagging

Name: Sachin Mysore Satish
Email id: smsatish@usc.edu

In this assignment, I designed the implementation of a sequence-to-sequence RNN model in TensorFlow.

1. Build_inference function:
In the build_inference function where we create logits, the following are the parameters' values that helped me achieve the accuracy of 95.6 in Italian and 95.1 in Japanese and with > 90% percentile in the secret language.

state_size(S) = 70
embedding_size(E) = 150
learning_rate(LR) = 0.01
batch_size(B) = 25

Initially, I had used the Simple RNN Cell with the tf.contrib.rnn.RNNCell which was giving me an accuracy of ~ 92% in Japanese language and ~ 93% in the Italian languages(S = E = 100, LR = 0.0055, B=32).

Steps performed to improve the accuracy:

a. Initially I tried to improve the accuracy by changing the learning rates and the state_size parameters and after much trials, I was able to reach the accuracy of 94.5 % in Italian and 94.3 % in Japanese.

b. To increase the accuracy even more further, I decided to use the tf.contrib.rnn.LSTMCell with tf.nn.bidirectional_dynamic_rnn so that the input sequence will be processed in both the forward and the backward ways giving richer representation and capturing more patterns. This resulted in increasing the accuracy close to 95% in Italian and 94.7% in Japanese.

EMBEDDING_SIZE = 100		BATCH_SIZE = 32		
State_size	Learning_rate	TRAIN_TIME_MIN	Italian Accuracy	Japanese Accuracy
75	0.007	None	94.9	94.6
50	0.007	None	95.1	94.5
50	0.007	9	95.2	94.7
50	0.007	10	95.1	94.3
50	0.008	9	95.1	94.7
50	0.0075	9	94.9	94.7
40	0.007	9	95.1	94.7
70	0.01	9	95.1	94.5
60	0.01	9	95.06	94.6

c. Sometimes the program was getting killed while it was running for Italian languages and setting the TRAIN_TIME_MINUTES = 9 stopped the process kill. Later, by reducing the batch_size, I started experimenting on the other parameters. Setting TRAIN_TIME_MINUTES = 9, State_size = 70, decreasing the batch_size to 25 and increasing the embedding_size to 150(approximately double of state_size) helped me to receive the accuracy of 95.6% in Italian and 95.1% in Japanese with secret language percentile over 90%.

b. build_training:

1) Here, tf.contrib.seq2seq.sequence_loss was used and added the loss computed from above using tf.losses.add_loss(loss) function.

2) I used the AdamOptimizer over the basic stochastic gradient descent which also helped in achieving better accuracy.

Please find the implementation of build_training below with comments:

```
# TODO(student): You must implement this.
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.
    """
    #Call to convert the vectors to binary matrix( tf.sequence mask is used)
    var = self.lengths_vector_to_binary_matrix(self.lengths)

    # Compute the loss using sequence loss and add to total loss
    loss = tf.contrib.seq2seq.sequence_loss(self.logits, self.targets, var, softmax_loss_function=None)
    tf.losses.add_loss(loss)

    # Update the network weights using Adam optimizer
    opt = tf.train.AdamOptimizer(learning_rate=0.01)

    #Creates an operation that evaluates the gradients and returns the loss
    self.train_op = tf.contrib.training.create_train_op(loss, opt)

    #Initialize the session and global variables
    self.sess = tf.Session()
    self.sess.run(tf.global_variables_initializer())
    pass
```