Natural Language Processing – Deep Learning Coding Assignment 2
Sequence to Sequence RNN for POS tagging

Neha Shivaprasad <nshivapr@usc.edu>

In this pdf explanation needed, I will be explaining my approach to this assignment based on the starter code and the changes I made to the starter code to enhance my accuracy on Italian and Japanese Languages, including making the model robust to make sure it works well on secret language as well. I will also be explaining the important functions in the class **SequenceModel** with comments, pictures and code snippets.

1. run inference: Given sentences (i.e. matrices of term IDs and their lengths), I return the part of speech tag ID for every word in every sentence

2. build_training: tf.contrib.seq2seq.sequence_loss is the loss function used as given by the hints. Logits, a tensor which is 3 dimensional, that is batch_size * maximum_length * number_of_tags . targets, for every word in every sentence, target: indicates the tag we want to predict into. Weights: weighing every element in our prediction, I give each of the targets a separate weight. When using weights as masking, I set all the valid timesteps to 1 and all the added timesteps to 0, like a mask returned by the tf.sequence.mask() as in the lengths_vector_to_binary_matrix function. I use Adam optimizer for training the model and initialize the session in this function.

```
def build_training(self):
    weights = self.lengths_vector_to_binary_matrix(self.lengths)

    # Computes the log likelihood of the tag sequence
    tf.losses.add_loss(tf.contrib.seq2seq.sequence_loss(self.logits, self.target,
weights))

    # Adam Optimizer
    opt = tf.train.AdamOptimizer(learning_rate=self.learning_rate)

    # Minimizing the total losses which also includes the regularization loss
computed using fully connected layer
    self.train_op = tf.contrib.training.create_train_op(tf.losses.get_total_loss(),
opt)

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

3. Build_inference: tf.nn.embedding_lookup, it takes 2 tensors, first being the floating point tensor(dimension being embeddings) that is the number of terms, second argument is a integer tensor, every sentence is a row padded with zeros, int64, it gives a tensor with one dimension added to it,

batch_size, max_length and added dimension is ‘d’. I use the loss function to train our model, finally I map this to the dimensions we care about. ‘d’ will become the number of tags. Here I use LSTM with Bidirectional RNN. The output of Bidirectional_dynamic_rnn is a tuple (lstm_forward, lstm_backward) containing the forward and backward rnn output Tensor.

```
def build_inference(self):
    hidden_size = 70
    embedding_size = 150

    embeddings = tf.get_variable('embeddings', shape=[self.num_terms, embedding_size])
    xembedding_matrix = tf.nn.embedding_lookup(embeddings, self.x)

    lstm_forward_cell = tf.nn.rnn_cell.LSTMCell(hidden_size)  # forward direction cell
    lstm_backward_cell = tf.nn.rnn_cell.LSTMCell(hidden_size)  # backward direction cell

    (lstm_forward, lstm_backward), _ = tf.nn.bidirectional_dynamic_rnn(lstm_forward_cell,
lstm_backward_cell,
                                                       xembedding_matrix,
                                                       dtype=tf.float32)
    output_rnn = tf.concat([lstm_forward, lstm_backward], axis=-1)

    self.logits = tf.layers.dense(output_rnn, units=self.num_tags, activation=None)
```

Accuracy:

Italian – 95.1%
Japanese – 94.9%
Secret Language : >90th percentile

Hyper parameters:

batch_size = 25
state_size = 70 (approximately equal to the number of tags)
embedding_size = 150
learning_rate = 0.01

I had implemented SimpleRNN using Tensorflow as given in the Hints, that is tf.keras.layers.SimpleRNNCell which gave an accuracy of 88% for Italian and 90% for Japanese. Post this I moved on to implement Bidirectional RNN with LSTM (Long Short Term Memory) in order to achieve better accuracy. I kept changing the state size and embedding size proportionally, increasing both of them led to higher accuracy.