Sequence to Sequence RNN for POS Tagging

CSCI 544 Final Deep learning Report

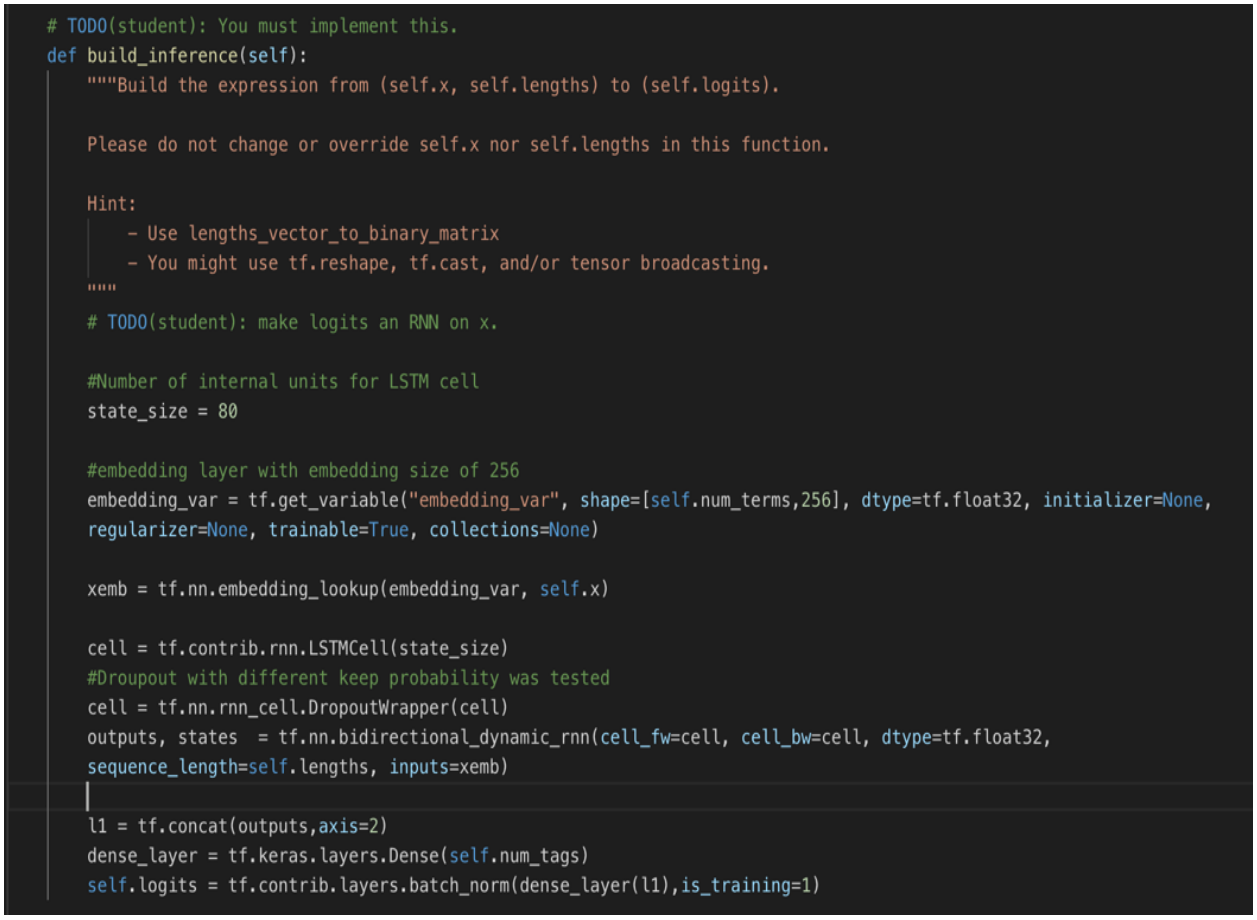
Shree Devi Badaja Narayan | badajana@usc.edu

Model:

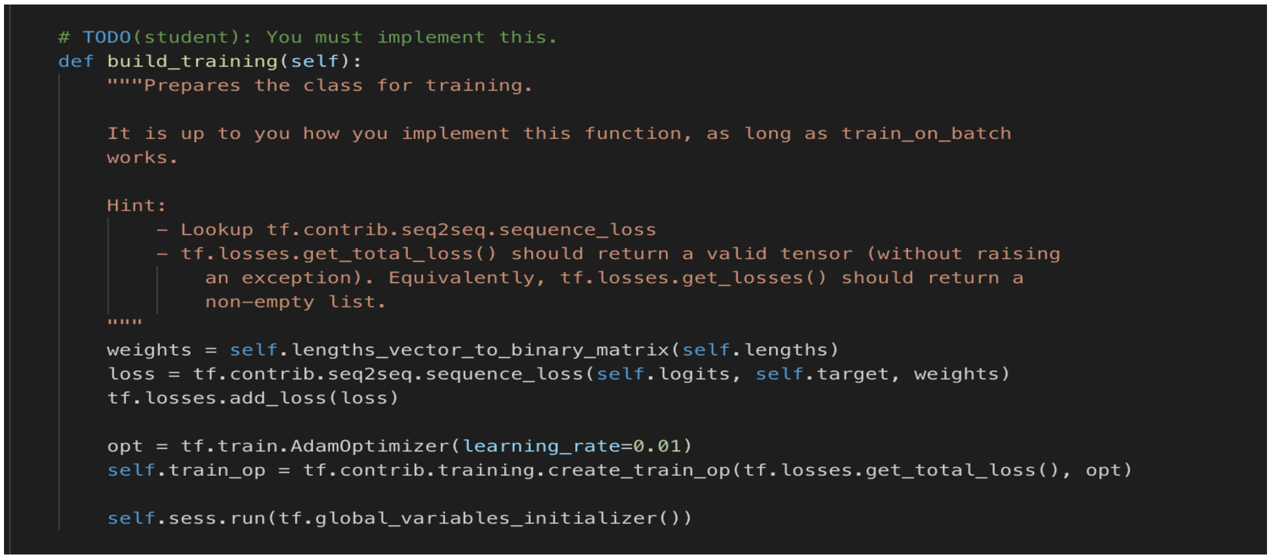
The objective of this project was POS tagging using Sequence to Sequence RNN. The implemented model consists of multiple layers; Embedding Layer followed by a Bi-Directional LSTM Layer with a dropout value and a Dense Layer. This is then followed by a batch normalization layer. Seq2Seq loss was implemented for the model.

Final Accuracy achieved with the above model for the Italian language was 95.5% , Japanese 95% and secret language > 95th .

A detailed implementation of the training can be found below



```
# TODO(student): You must implement this.
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.

    #Number of internal units for LSTM cell
    state_size = 80

    #embedding layer with embedding size of 256
    embedding_var = tf.get_variable("embedding_var", shape=[self.num_terms,256], dtype=tf.float32, initializer=None,
    regularizer=None, trainable=True, collections=None)

    xemb = tf.nn.embedding_lookup(embedding_var, self.x)

    cell = tf.contrib.rnn.LSTMCell(state_size)
    #Droupout with different keep probability was tested
    cell = tf.nn.rnn_cell.DropoutWrapper(cell)
    outputs, states  = tf.nn.bidirectional_dynamic_rnn(cell_fw=cell, cell_bw=cell, dtype=tf.float32,
    sequence_length=self.lengths, inputs=xemb)

    l1 = tf.concat(outputs,axis=2)
    dense_layer = tf.keras.layers.Dense(self.num_tags)
    self.logits = tf.contrib.layers.batch_norm(dense_layer(l1),is_training=1)
```



```
    # 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.
        """
        weights = self.lengths_vector_to_binary_matrix(self.lengths)
        loss = tf.contrib.seq2seq.sequence_loss(self.logits, self.target, weights)
        tf.losses.add_loss(loss)

        opt = tf.train.AdamOptimizer(learning_rate=0.01)
        self.train_op = tf.contrib.training.create_train_op(tf.losses.get_total_loss(), opt)

        self.sess.run(tf.global_variables_initializer())
```

Tuning of Hyper Parameters:

Multiple combinations of the hyper parameters were experimented to reach the highest accuracy. The best one has been captured below

- Batch size = 32
- Learning rate of the Adam optimizer = 0.01
- Embedding size = 256
- State size (hidden size) of LSTM cell = 80

Models Tried and Tested:

- SimpleRNNCell followed by a keras dense layer was the initial model tried. Attained an accuracy of (90-92%). Tested the model with a learning rate of 0.05 and 0.01 which reduced the accuracy. Also tried out different batch sizes such as 64. Maximum accuracy was attained with embedding size 80, state size 70, learning rate 0.01 and batch size 64. Increasing embedding size from 40 to 80 improved the accuracy in Japanese language.
- Second Model tested was with Keras Bidirectional LSTM followed by a TimeDistributed Dense layer. This improved the accuracy to around 93%. Added batch normalization layer to improve the accuracy to 94%. Adding dropout to the LSTM layer did not give any noticeable difference.
- Final model tried was a Bi-Directional LSTM Layer with a dropout value and a Dense Layer, followed by a batch normalization with the hyper parameters as described.
  Bi-Directional LSTM Layer increased the accuracy by 0.5 % and batch normalization improved it by another 0.5%. Increasing the embedding size from 80, 100, 125 , 256 produced a significant improvement in Japanese accuracy from without affecting Italian.
  The above model significantly improved the Japanese accuracy from 94% to 95%.

Final accuracy reached was Italian language was 95.5% , Japanese 95% and secret language > 95th