# POS Tagging Using Bidirectional Dynamic RNN

Natural Language Processing
Rakshanda Agarwal
rakshana@usc.edu

## Introduction:

The Goal of this assignment was the prediction of POS tags for words in any language provided the model is trained on that language. This model has been implemented using a bi-directional recurrent neural network followed by a fully connected layer.

## Function Implementations:

```
def build_inference(self):
    #Dynamic RNN
    state_size = 100#60#70#60#50#32
    emsize = 100#30#10
    #Tried experimenting with different state_size and embedding_size for best
    results
    embeddings = tf.get_variable('embeddings',[self.num_terms, emsize])
    xembed = tf.nn.embedding_lookup(embeddings, self.x)
    #Create an embedding lookup of training data
    cell_fw = tf.contrib.rnn.LSTMCell(state_size)
    cell_bw = tf.contrib.rnn.LSTMCell(state_size)
    #Create two RNN LSTM cells
    (of,ob),_ = tf.nn.bidirectional_dynamic_rnn (cell_fw,cell_bw,xembed,
                sequence_length=self.lengths,dtype=tf.float32)
    dense = tf.concat([of,ob],axis=-1)
    #Concatenate both forward and backward outputs
    self.logits = tf.contrib.layers.fully_connected(dense,
                 int(self.num_tags),activation_fn=None)

def build_training(self):
    weights = self.lengths_vector_to_binary_matrix(self.lengths)
    loss = tf.contrib.seq2seq.sequence_loss(logits=self.logits,
           targets=self.targets,weights=weights)
    #Create seq2seq loss and add it to total losses
    tf.losses.add_loss(loss)
    opt = tf.train.AdamOptimizer(self.lr)
    self.train_opt = tf.contrib.training.create_train_op
                     (tf.losses.get_total_loss(), opt)
```

```
#Use AdamOptimizer for training
self.sess.run(tf.global_variables_initializer())
```

## Hyper Parameters:

- Batch size=32
- Embedding size=100
- State size=100
- Learning Rate=0.008
  With decay in each epoch
  `self.lr *= (0.008/(2**self.epoch))`

## Observations and Trials:

On trying different models and parameters, these are the things I observed.

- SimpleRNNCell was slower in computation and gave a lower accuracy (90-92%).
- Lower Learning rate with decay gave better results. I initially started with 0.05 and then moved down to 0.008 where I could see satisfactory results.
- Early stopping did not help as Japanese and Italian datasets varied vastly.
- I tried combining RNN and LSTM, but again time constraints did not let it train completely.
- Adding multiple layers gave no noticeable difference.
- I also tried Densely connected RNN which showed improvement in accuracy but only with a greater train time.
- On increasing the batch size, training was faster but the accuracy dropped 3-4 epochs.

## Results:

The model was trained for about 4-5 epochs due to time constraints but gave a high accuracy of 95.8% on Italian dataset and 95.3% on Japanese dataset. While on the hidden language, the accuracy was in the top 95 percentile submissions.