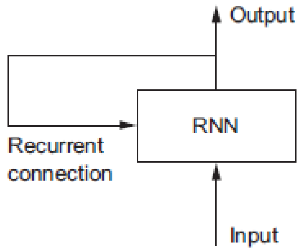
Seq2Seq Recurrent Neural Network for POS Tagging

Vaibhav Jain

May 2019

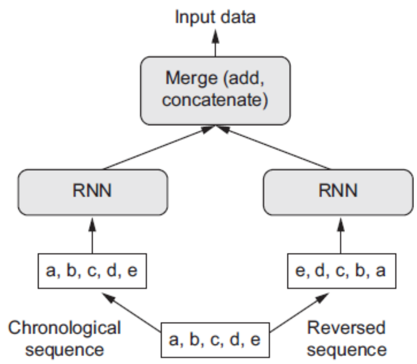
Introduction

A recurrent neural network processes inputs by iterating through the sequence elements and maintaining a *state* containing information relative to what it has seen so far.



Bidirectional RNN

A bidirectional RNN processes the input sequence both ways, obtaining potentially richer representations and capturing patterns that may have missed by a Simple RNN.



Code (Accuracy 95.3 in Japanese, 95.7 in Italian)

```
def bidirectional_rnn():
    state_size = 104
    embedding_size = 52

    # embeddings = [batch_size X embedding_size]
    embeddings = tf.get_variable('embeddings', [self.num_terms, embedding_size])

    # RNN Input
    # inputs =  [batch_size X max_length X embedding_size]
    inputs = tf.nn.embedding_lookup(embeddings, self.x)

    """
    Bidirectional LSTM Layer
    It reads the input from left to right and right to left and gives
    2 outputs (output forward and output backward) which are concatenated
    together to form the final output.
    tf.contrib.rnn.stack_bidirectional_rnn can be used instead of
tf.nn.bidirectional_dynamic_rnn
    The later one is costlier but more efficient
    """

    # Create LSTM cell
    rnn_cell = tf.nn.rnn_cell.LSTMCell(state_size)

    # output_fw and output_bw = [batch_size X max_length X state_size]
    (output_fw, output_bw), output_states = tf.nn.bidirectional_dynamic_rnn(
        cell_fw=rnn_cell,
        cell_bw=rnn_cell,
        inputs=inputs,
        sequence_length=self.lengths,
        dtype=tf.float32)
    inputs = tf.concat([output_fw, output_bw], axis=-1)
```

```
# Dense Layer with Linear activation Function(default)
logit_inputs = tf.reshape(inputs, [-1, 2 * state_size])
self.logits = tf.layers.dense(logit_inputs, self.num_tags)
self.logits = tf.reshape(self.logits, [-1, self.max_length, self.num_tags])
```

Hyper parameters
batch_size = 32
state_size *(for RNN cell)* = 104
embedding_size = 52
learning_rate = 0.*0175 (decreasing by a factor of 2 on each epoch)*

Insights
1. Initially I implemented SimpleRNN using tf.keras.layers.SimpleRNNCell which led to an accuracy of ~90 to 92%.

2. Afterwards, I stacked more than one RNN (Simple/LSTM/GRU) Layer to increase the representational power of the network. There was no noticeable difference in the accuracy in comparison with a 1-Layer RNN.

3. To deal with vanishing gradient problem of Simple RNN, I used gradient clipping which again did not help much in our case.

4. Later, I moved on to Bidirectional RNN which can catch patterns that may be overlooked by a unidirectional RNN. It led to ~95 to 96% accuracy in both the languages.

5. To get high accuracy, the state and embedding size were kept around 100-120 and 50-60 units respectively. Increasing both of them further led to an increase in accuracy but it needed more train time. Setting TRAIN_TIME_MINUTES > 8 was killing the process for Italian. So the state and embedding size were not increased further.

6. I tried several learning rates ranging from 0.001 to 0.02. Keeping it as 0.0175 and decreasing it by a factor of 2 on each epoch led to a constant accuracy of 95% or more in both Japanese and Italian.

7. Using gradient clipping helped achieving a better accuracy in the secret language.