I. **Unidirectional RNN**

Converged Loss: 5.191

2017-11-02 22:18:03 INFO: At 0-th epoch.
2017-11-02 22:21:45 INFO: Average loss value per instance is 5.78555583954 at the end of epoch 0
2017-11-02 22:21:45 INFO: At 1-th epoch.
2017-11-02 22:25:07 INFO: Average loss value per instance is 5.53667545319 at the end of epoch 1
2017-11-02 22:25:07 INFO: At 2-th epoch.
2017-11-02 22:28:33 INFO: Average loss value per instance is 5.40552377701 at the end of epoch 2
2017-11-02 22:28:33 INFO: At 3-th epoch.
2017-11-02 22:31:55 INFO: Average loss value per instance is 5.35254621506 at the end of epoch 3
2017-11-02 22:31:55 INFO: At 4-th epoch.
2017-11-02 22:35:35 INFO: Average loss value per instance is 5.29030370712 at the end of epoch 4
2017-11-02 22:35:35 INFO: At 5-th epoch.
2017-11-02 22:39:50 INFO: Average loss value per instance is 5.25883102417 at the end of epoch 5
2017-11-02 22:39:50 INFO: At 6-th epoch.
2017-11-02 22:43:47 INFO: Average loss value per instance is 5.23208522797 at the end of epoch 6
2017-11-02 22:43:47 INFO: At 7-th epoch.
2017-11-02 22:47:46 INFO: Average loss value per instance is 5.2171163559 at the end of epoch 7
2017-11-02 22:47:46 INFO: At 8-th epoch.
2017-11-02 22:51:41 INFO: Average loss value per instance is 5.19942140579 at the end of epoch 8
2017-11-02 22:51:41 INFO: At 9-th epoch.
2017-11-02 22:55:38 INFO: Average loss value per instance is 5.19164133072 at the end of epoch 9
2017-11-02 22:55:38 INFO: Early stopping triggered with threshold 9 (previous dev loss: 5.19942140579, current: 5.19164133072)

II. **Bidirectional RNN**

- While implementing a bidirectional rnn, we just added a new layer that computes the hidden information from right to left. The architecture is as following:

$$h_{tf} : \textit{forward output vector}$$
$$h_{tb} : \textit{backward output vector}$$

$y : final\ output\ vector$

$a_1 : tanh$

$a_2 : sigmoid\ activation$

$W, V\ U\ and\ b\ : Weight\ matrices\ and\ the\ bias.$

$h_{tf}\ :\ a_1(W * x_t\ +\ V * x_{t-1}\ +\ b)$

$h_{tb}\ :\ a_1(W * x_t\ +\ V * x_{t+1}\ +\ b)$

$y_t\ : a_2\ (U * [h_{tf}; h_{tb}]\ +\ c)$

We concatenated the two hidden outputs - forward and backward and that was then given as the final output to the sigmoid function. This helped include the left and the right context information for each word.

Converged Loss : 4.277
2017-11-05 01:35:01 INFO: At 0-th epoch.
2017-11-05 01:38:18 INFO: Average loss value per instance is 4.89027929306 at the end of epoch 0
1
2017-11-05 01:38:18 INFO: At 1-th epoch.
2017-11-05 01:41:32 INFO: Average loss value per instance is 4.50841760635 at the end of epoch 1
2
2017-11-05 01:41:32 INFO: At 2-th epoch.
2017-11-05 01:44:47 INFO: Average loss value per instance is 4.33946371078 at the end of epoch 2
3
2017-11-05 01:44:47 INFO: At 3-th epoch.
2017-11-05 01:48:00 INFO: Average loss value per instance is 4.27752733231 at the end of epoch 3
4
2017-11-05 01:48:00 INFO: At 4-th epoch.
2017-11-05 01:51:15 INFO: Average loss value per instance is 4.2897310257 at the end of epoch 4
2017-11-05 01:51:15 INFO: Early stopping triggered with threshold 4 (previous dev loss: 4.27752733231, current: 4.2897310257)

III. Multi-word cloze:

a. The accuracy obtained using a simple bi-directional rnn is 0.258.
b. As an improvement to the bi-directional model we implemented a GRU.

A GRU (gated recurrent unit) are just a gating mechanism in a recurrent network. The architecture of a GRU is as following:

$$
\begin{aligned}
&xt \ : \ input\ vector \\
&ht \ : \ output\ vector \\
&zt \ : \ update\ gate\ vector \\
&rt \ : \ reset\ gate\ vector \\
&W, U\ and\ b \ : \ weight\ matrices\ and\ the\ bias\ vector. \\
&a1 \ : \ sigmoid\ function\ activation \\
&a2 \ : \ tanh\ activation \\
&zt = a_1(W_z * x_t + U_z * h_{t-1} + b_z) \\
&rt = a_1(W_t * x_t + U_r * h_{t-1} + b_r) \\
&ht = z_t * h_{t-1} + (1 - z_t) * a_2(W_h * x_t + U_h * (r_t * h_{t-1} + b_h)
\end{aligned}
$$

The final performance with the Bi GRU model is 0.396 with the converged loss at 3.55.

The results are generated after trying several hyperparameters. Several combinations gave similar performance, but the one with the best performance was
Learning Rate: 1e-3
Batch Size: 32
Embedding Size: 512 (256 works nearly as good as well)
Recurrent Size: 128

A lower learning rate ensures that the model doesn't overfit for a larger recurrent layer and the training loss doesn’t jitter (variance) as the model converges. A lower batch size ensures faster convergence. However it needs to be high enough to reduce training time and prevent large variance.