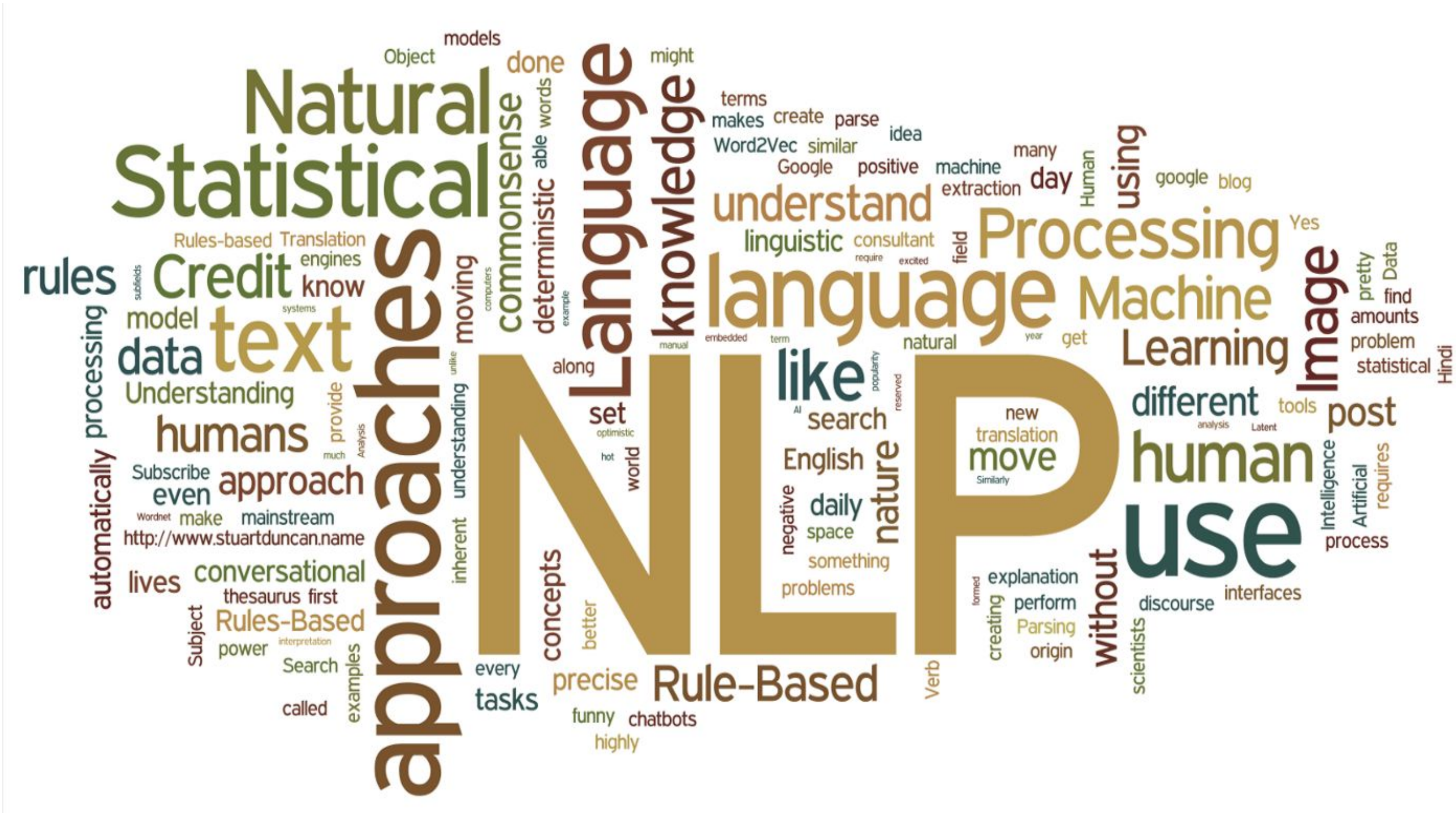
NLA: Natural Language Processing Applications

Fine Tune BERT for Extractive Summarization

Names (Roll Number):
Aman Krishna (2018201070)
Ishaan Tyagi (2018201017)
Rajat Yadav (2018201077)



Introduction: Extractive Summarization

Single-document summarization is the task of automatically generating a shorter version of a document while retaining its most important information.

The task is often divided into two paradigms, abstractive summarization and extractive summarization. In abstractive summarization, target summaries contain words or phrases that were not in the original text and usually require various text rewriting operations to generate, while extractive approaches form summaries by copying and concatenating the most important spans (usually sentences) in a document. Our focus is on extractive summarization using the pre-trained model BERT.

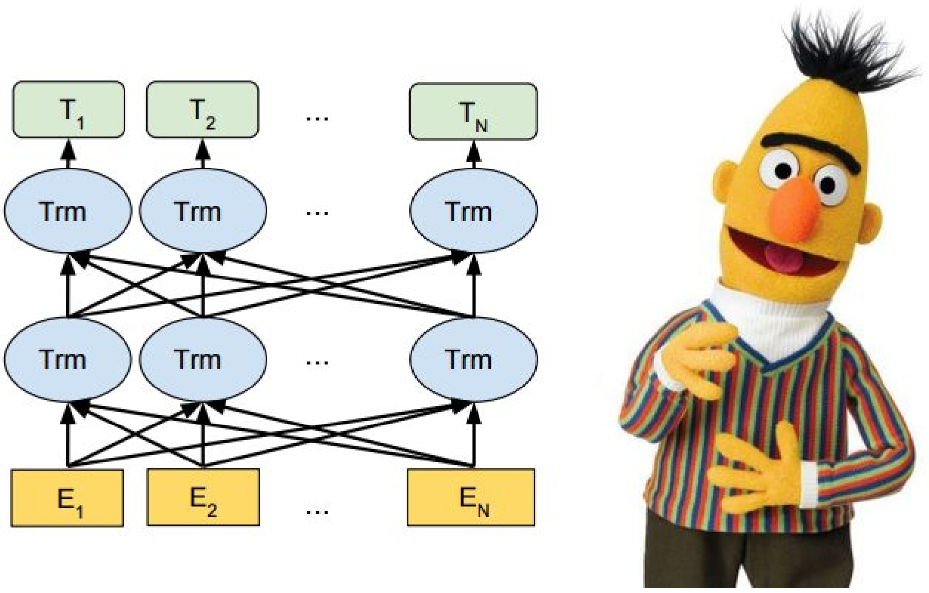


Fig: BERT Model

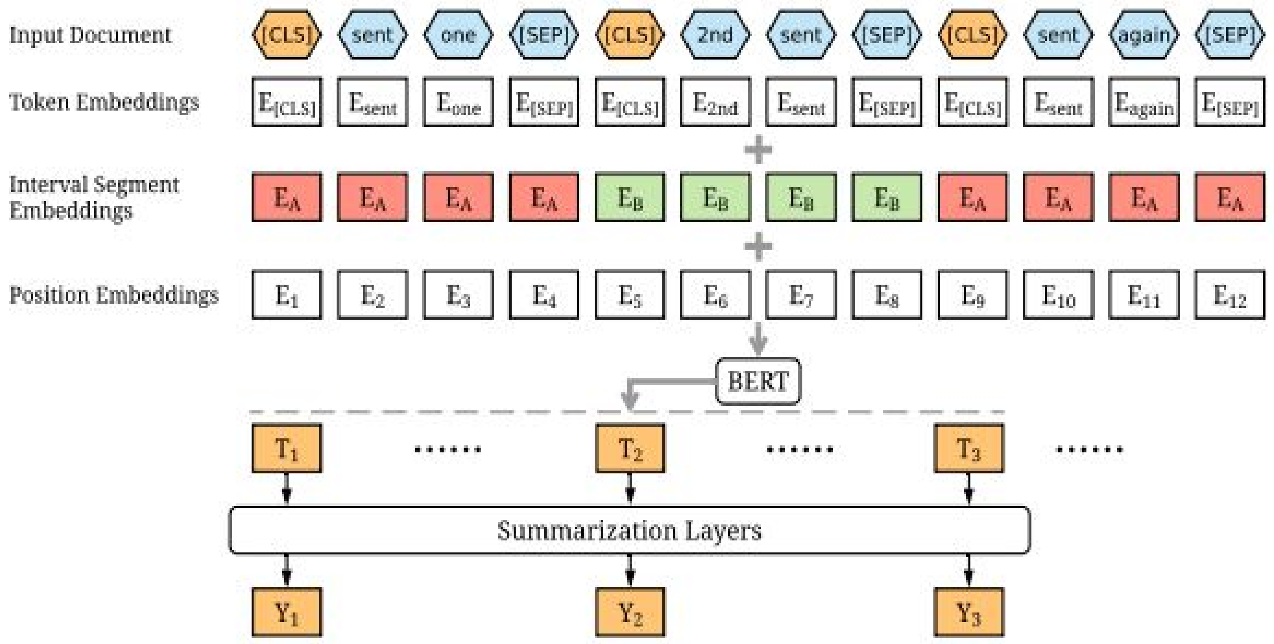
Methodology: How we do what we do!

Let **d** denote a document containing several sentences [***sent*** $_1$, ***sent*** $_2$, ... , ***sent*** $_m$], where ***sent*** $_i$ is the i-th sentence in the document.

Extractive summarization can be defined as the task of assigning a label

$$y_i \in \{0, 1\}$$

to each ***sent*** $_i$, indicating whether the sentence should be included in the summary. It is assumed that summary sentences represent the most important content of the document. The following diagram shows the steps:



- **[CLS] & [SEP]:** Special symbol inserted at the start and end of the sentence

- **E_X:** Token embedding for word X
- **E_A & E_B:** Embedding for even and odd sentences
- **E_i:** Positional Embeddings
- **T_i:** Output Embeddings of BERT
- **Y_i:** Output for each sentence i.e. whether to keep or reject

Case I: Simple Classifier

Simple Classifier Like in the original BERT paper, the Simple Classifier only adds a linear layer on the BERT outputs and uses a sigmoid function for prediction.

$$\hat{Y}_i = \sigma(W_o T_i + b_o)$$

Case II: Intersentence Transformer

Instead of a simple sigmoid classifier, Inter-sentence Transformer applies more Transformer layers only on sentence representations, extracting document-level features focusing on summarization tasks from the BERT outputs:

$$\tilde{h}^l = \mathrm{LN}(h^{l-1} + \mathrm{MHAtt}(h^{l-1}))$$
$$h^l = \mathrm{LN}(\tilde{h}^l + \mathrm{FFN}(\tilde{h}^l))$$

where

- T: Sentence vectors output by BERT
- h_0 = PosEmb(T), where PosEmb is the function of adding positional embeddings (indicating the position of each sentence) to T
- LN: Layer normalization operation
- MHAtt: Multi-head attention operation
- Superscript l: The depth of the stacked layer
- Final output layer is still the Sigmoid layer

Case III: BERT + LSTM

We apply an LSTM layer over the BERT outputs to learn summarization-specific features. At time step i, the input to the LSTM layer is the BERT output T_i , and the output is calculated as:

$$\begin{pmatrix} F_i \\ I_i \\ O_i \\ G_i \end{pmatrix} = \mathrm{LN}_h(W_h h_{i-1}) + \mathrm{LN}_x(W_x T_i)$$

$$\begin{aligned} C_i = {} & \sigma(F_i) \odot C_{i-1} \\ & + \sigma(I_i) \odot \tanh(G_{i-1}) \\ h_i = {} & \sigma(O_t) \odot \tanh(\mathrm{LN}_c(C_t)) \end{aligned}$$

- **F_i , I_i , O_i:** Forget gates, Input gates, Output gates
- **G_i:** Hidden vector
- **C_i:** Memory vector
- **h_i:** Output vector
- **LN_h, LN_x, LN_c:** Different Layer Normalization operations

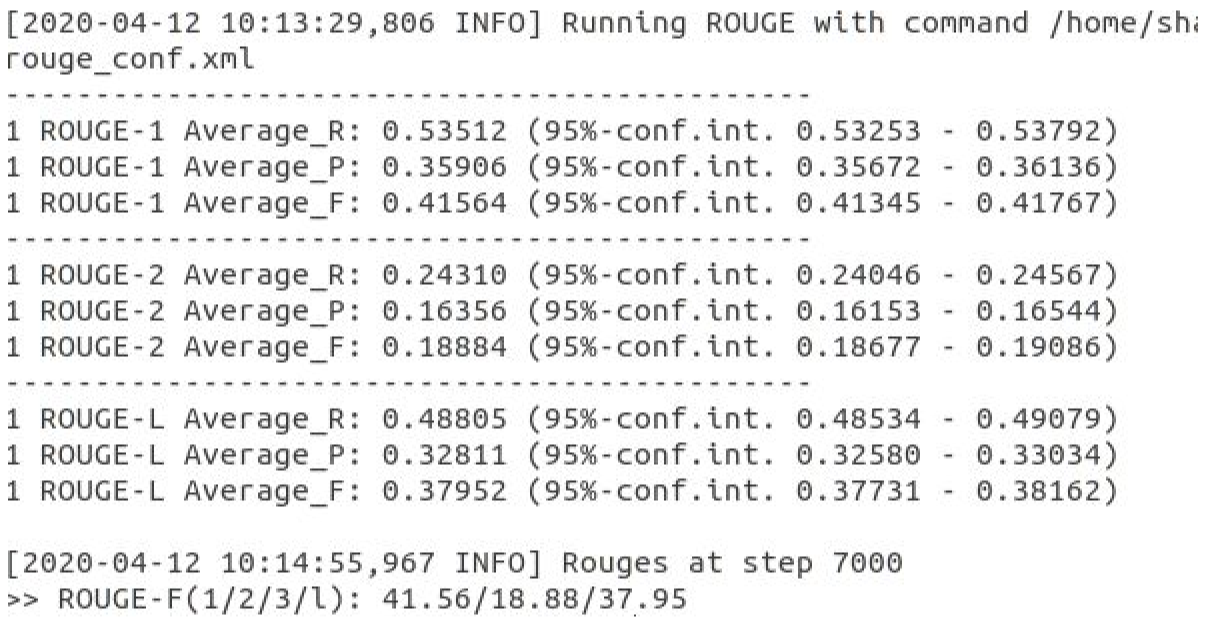
Implementation:

- **Dateset:** We evaluated on two benchmark datasets, namely the CNN/DailyMail news highlights dataset and the New York Times Annotated Corpus.
- **Library:** We used PyTorch, OpenNMT and the ‘bert-base-uncased’∗ version of BERT to implement the model. BERT and summarization layers are jointly fine-tuned
- **Solver:** Adam with β1 = 0.9, β2 = 0.999 is used for fine-tuning
- **Iterations:** The paper suggested running the experiment for 50000 iterations. However, due to lack of similar resources (as mentioned in paper) we ran the experiment for 10000 iterations and the results obtained suggest we would achieve similar results by running it for the full duration
- **Batch Size:** We reduced batch size from 3000 to 1000 as it overflowed our GPU VRAM
- **GPU:** The paper used three GTX 1080ti for their experiment. We have used a single GPU (GTX 1070m: Laptop GPU)

Results:

After running the tests the following results were obtained:

- **Case I: Simple Classifier using Sigmoid**



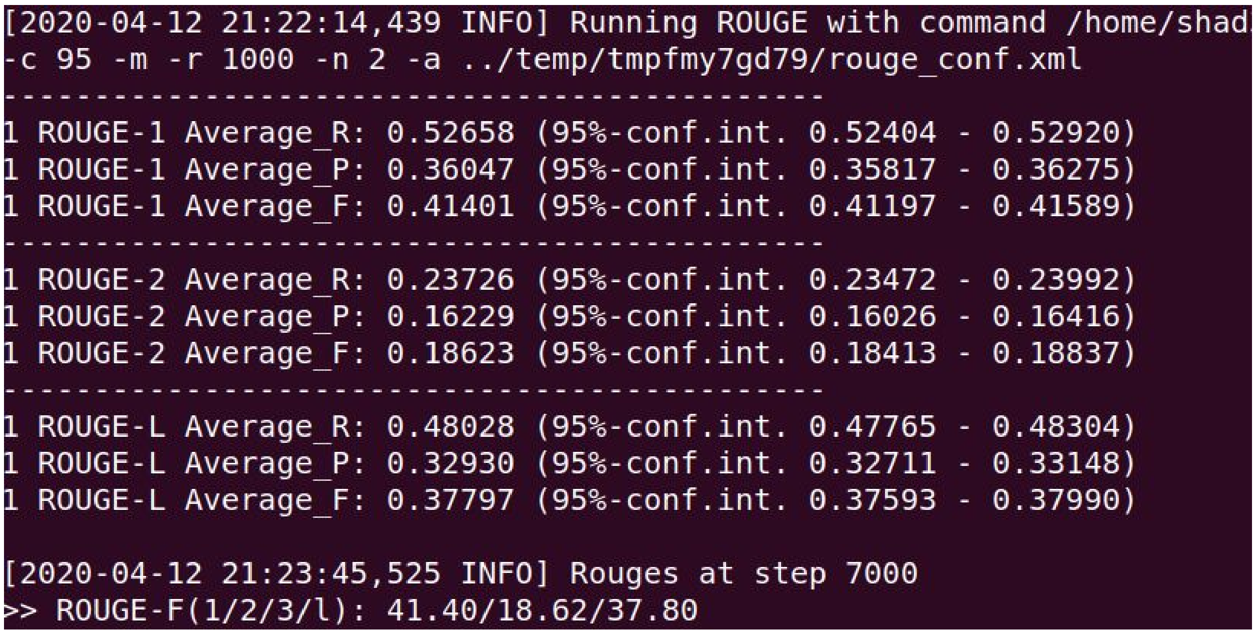
```
[2020-04-12 10:13:29,806 INFO] Running ROUGE with command /home/sh
rouge_conf.xml
---------------------------------------------
1 ROUGE-1 Average_R: 0.53512 (95%-conf.int. 0.53253 - 0.53792)
1 ROUGE-1 Average_P: 0.35906 (95%-conf.int. 0.35672 - 0.36136)
1 ROUGE-1 Average_F: 0.41564 (95%-conf.int. 0.41345 - 0.41767)
---------------------------------------------
1 ROUGE-2 Average_R: 0.24310 (95%-conf.int. 0.24046 - 0.24567)
1 ROUGE-2 Average_P: 0.16356 (95%-conf.int. 0.16153 - 0.16544)
1 ROUGE-2 Average_F: 0.18884 (95%-conf.int. 0.18677 - 0.19086)
---------------------------------------------
1 ROUGE-L Average_R: 0.48805 (95%-conf.int. 0.48534 - 0.49079)
1 ROUGE-L Average_P: 0.32811 (95%-conf.int. 0.32580 - 0.33034)
1 ROUGE-L Average_F: 0.37952 (95%-conf.int. 0.37731 - 0.38162)

[2020-04-12 10:14:55,967 INFO] Rouges at step 7000
>> ROUGE-F(1/2/3/l): 41.56/18.88/37.95
```

Linear Classifier	**Our Result**	**Paper's Result**
ROUGE-1	41.56	43.23
ROUGE-2	18.88	20.22
ROUGE-L	37.95	39.6

- **ROUGE-N:** measures unigram, bigram, trigram and higher order n-gram overlap
- **ROUGE-L:** measures the longest matching sequence of words using LCS

- **Case II: Intersentence Transformer**



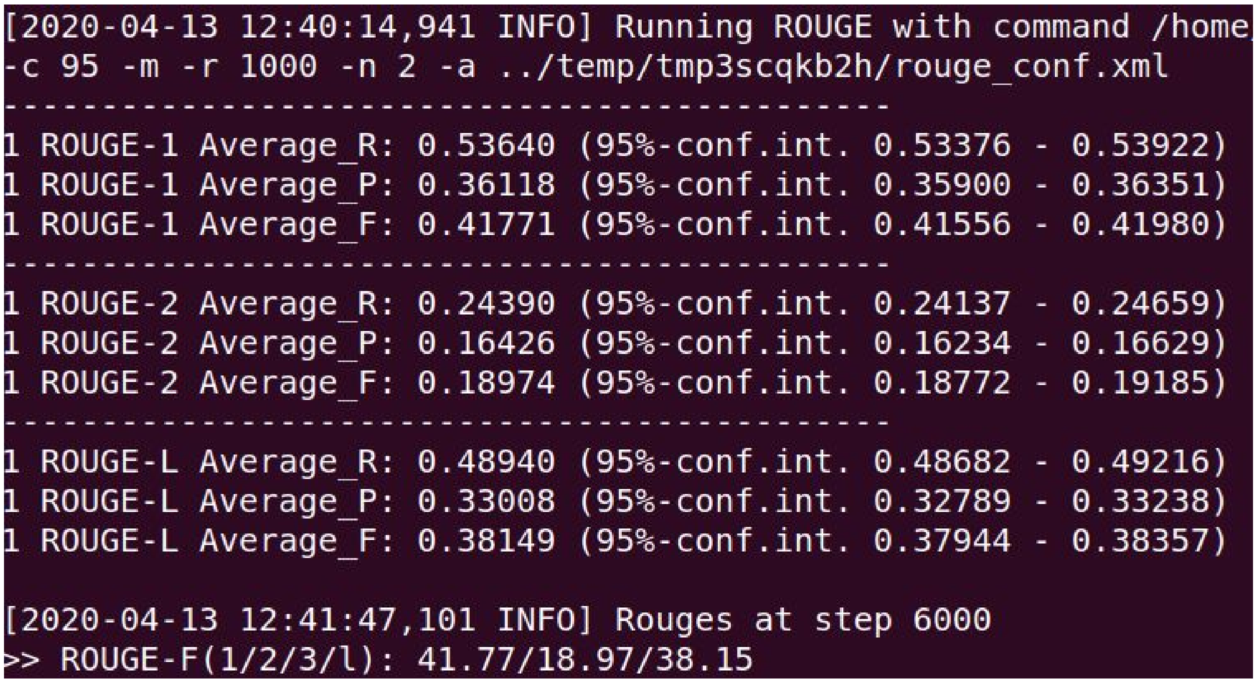
```
[2020-04-12 21:22:14,439 INFO] Running ROUGE with command /home/shad
-c 95 -m -r 1000 -n 2 -a ../temp/tmpfmy7gd79/rouge_conf.xml
---------------------------------------------
1 ROUGE-1 Average_R: 0.52658 (95%-conf.int. 0.52404 - 0.52920)
1 ROUGE-1 Average_P: 0.36047 (95%-conf.int. 0.35817 - 0.36275)
1 ROUGE-1 Average_F: 0.41401 (95%-conf.int. 0.41197 - 0.41589)
---------------------------------------------
1 ROUGE-2 Average_R: 0.23726 (95%-conf.int. 0.23472 - 0.23992)
1 ROUGE-2 Average_P: 0.16229 (95%-conf.int. 0.16026 - 0.16416)
1 ROUGE-2 Average_F: 0.18623 (95%-conf.int. 0.18413 - 0.18837)
---------------------------------------------
1 ROUGE-L Average_R: 0.48028 (95%-conf.int. 0.47765 - 0.48304)
1 ROUGE-L Average_P: 0.32930 (95%-conf.int. 0.32711 - 0.33148)
1 ROUGE-L Average_F: 0.37797 (95%-conf.int. 0.37593 - 0.37990)

[2020-04-12 21:23:45,525 INFO] Rouges at step 7000
>> ROUGE-F(1/2/3/l): 41.40/18.62/37.80
```

BERT Transformer	Our Result	Paper's Result
ROUGE-1	41.40	43.25
ROUGE-2	18.62	20.24
ROUGE-L	37.80	39.63

- **ROUGE-N:** measures unigram, bigram, trigram and higher order n-gram overlap
- **ROUGE-L:** measures the longest matching sequence of words using LCS

Case III: BERT + LSTM:



```
[2020-04-13 12:40:14,941 INFO] Running ROUGE with command /home
-c 95 -m -r 1000 -n 2 -a ../temp/tmp3scqkb2h/rouge_conf.xml
---------------------------------------------
1 ROUGE-1 Average_R: 0.53640 (95%-conf.int. 0.53376 - 0.53922)
1 ROUGE-1 Average_P: 0.36118 (95%-conf.int. 0.35900 - 0.36351)
1 ROUGE-1 Average_F: 0.41771 (95%-conf.int. 0.41556 - 0.41980)
---------------------------------------------
1 ROUGE-2 Average_R: 0.24390 (95%-conf.int. 0.24137 - 0.24659)
1 ROUGE-2 Average_P: 0.16426 (95%-conf.int. 0.16234 - 0.16629)
1 ROUGE-2 Average_F: 0.18974 (95%-conf.int. 0.18772 - 0.19185)
---------------------------------------------
1 ROUGE-L Average_R: 0.48940 (95%-conf.int. 0.48682 - 0.49216)
1 ROUGE-L Average_P: 0.33008 (95%-conf.int. 0.32789 - 0.33238)
1 ROUGE-L Average_F: 0.38149 (95%-conf.int. 0.37944 - 0.38357)

[2020-04-13 12:41:47,101 INFO] Rouges at step 6000
>> ROUGE-F(1/2/3/l): 41.77/18.97/38.15
```

BERT + LSTM	**Our Result**	**Paper's Result**
ROUGE-1	41.77	43.22
ROUGE-2	18.97	20.17
ROUGE-L	37.15	39.59

Conclusions:

- Our setup falls a little short to the performance obtained in the paper due:
    - Reducing total training steps from 50000 to 10000
    - Maybe due to reducing batch size from 3000 to 1000
- The results of the three model were quite close may be because the output of BERT is rich enough that adding more transformer for inter-sentence relationships using transformers or by using LSTM little more details are added