Text Summarization with BERT

Ashish Kayastha

Department of Computer Science

Central Michigan University

1 Introduction

Text Summarization is an NLP task that deals with the creation of summary from a long piece of text. The summarized text should include several key ideas in the original text and should be coherent. There are basically two different approaches to text summarization in the literature:

a) Extractive Summarization
This approach selects the most important sentences in a long piece of text and shows them together with the same sequence of words.

b) Abstractive Summarization
This approach abstracts the key ideas in a long piece of text and generates new sentences that were not in the original text.

2 Background

Recently, pretrained language models have become popular for text summarization tasks. Originally, word embeddings such as Word2Vec [1,2] and GloVe [3] used to be very popular, but they generate the same vector representation for a particular word without any context. For instance, a sentence like "The CEO ran the marathon" would have the same token representation for 'ran' as the sentence "The CEO ran the company". Language models like Embeddings from Language Models (ELMO) [4] and Bidirectional Encoder Representations from Transformers (BERT) [5] tackles this issue by generating contextual representations from large-scale corpora. Historically, text summarization used to be done with sequence-to-sequence models with RNNs employed on both the encoder and the decoder. Lately, transformers [6] have supplanted RNNs as the dominant architecture of choice for implementing sequence-to-sequence summarization models. This is because transformers are more suitable for parallelization and have neural self-attention built into the architecture rather than as an afterthought.

3 Methods

3.1 Datasets

One of the benchmark datasets for text summarization is the CNN/DailyMail dataset that contains news articles and their highlights. To have comparable results with the previous methods, the standard splits of Hermann et al. [7] for train, validation, and test set were used. Table 1 shows the splits used as well as additional statistics from the dataset.

Dataset	# Docs (Train | Val | Test)	Avg. Doc Length (words | sentences)	Avg. Summary Length (words | sentences)
CNN	90,266 | 1,220 | 1,093	760.50 | 33.98	45.70 | 3.59
DailyMail	196,961 | 12,148 | 10,397	653.33 | 29.33	54.65 | 3.86
Total	287,227 | 13,368 | 11,490	706.91 | 31.65	50.17 | 3.72

Table 1. CNN/DailyMail Dataset Statistics

3.1 Preprocessing

The CNN/DailyMail dataset was first cleaned for any unnecessary tokens and the sentences were split with the Stanford CoreNLP [8] toolkit. Even though BERT can be used for most NLP tasks right out of the box, its application to text summarization task needs modification to the inputs. This is because BERT generates embeddings at token level rather than sentence level, which is a problem for multi-sentence documents. To tackle this issue, Liu et al. proposed the BERTSUM method specifically for text summarization [9]. This method proposes inserting external [CLS] tokens in the beginning of the sentence so that every [CLS] token generates representations for their corresponding sentences. In addition, they also propose having alternate interval segment embeddings to differentiate between the sentences depending on whether the sentence is odd or even.

Due to the problem of representing multi-sentence documents, the BERTSUM method was used for the preprocessing steps and extra [PAD] tokens were added if the source tokens were less than the maximum source length. Next, the BERT tokenizer from the Transformers library was used to generate the source token ids but generation of the necessary token and sentence masks were handled manually.

As for preprocessing of the target data, different steps were taken for extractive and abstractive models.

a) Extractive Preprocessing

To prepare the target data for training the extractive BERTSUM models, sentence labels were generated using a greedy algorithm, which maximize the ROUGE-2 score against the reference (gold) summaries.

b) Abstractive Preprocessing

To prepare the target data for training the abstractive BERTSUM-RNN models, [START] and [END] special tokens were inserted in the beginning and end of the reference (gold) summaries. Next, [SEP] tokens were inserted between the sentences for predicting the sentence boundaries and extra [PAD] tokens were added if the target tokens were less than the maximum target length. Finally, the BERT tokenizer from the Transformers library were used to generate the target token ids.

3.2 Training Environment

To train the extractive BERTSUM models and the abstractive BERTSUM-RNN models, Google Pro+ with TensorFlow and Keras stack were used. All the models were trained using NVIDIA V100-SXM2 graphical processing unit (GPU) hardware equipped with 16 GB of memory and 51 GB RAM.

3.3 Evaluation Metric

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [10] is a common set of metrics based on n-grams overlaps used for evaluating text summarization systems. The important ROUGE metrics for this study are:

a) ROUGE-N: N-gram Co-occurrence Statistics
   It is mainly used for assessing information.
   i. ROUGE-1
      It is the unigram recall between a candidate summary and a reference summary.

   ii. ROUGE-2
      It is the bigram recall between a candidate summary and a reference summary.

b) ROUGE-L: Longest Common Subsequence (LCS)
   It is the ratio of LCS between the candidate and reference summary to the total number of words in the reference summary. It is mainly used for assessing fluency.

To calculate these ROUGE evaluation metrics, pyrouge library was manually setup to work on Google Colab Pro+.

3.4 Training and Evaluation Procedure

a) Extractive Summarization
   For extractive summarization, the same BERTSUM [9] model architecture with fine-tuning transformer layers originally written in PyTorch by Liu et al. was implemented in

Keras. The model predicts the probability of each sentence to be part of the summary. All extractive models were trained with the same hyperparameters as BERTSUM for 5 epochs. Models were evaluated after every epoch and model checkpoints were saved as weights only if the validation loss was better than the previous epoch.

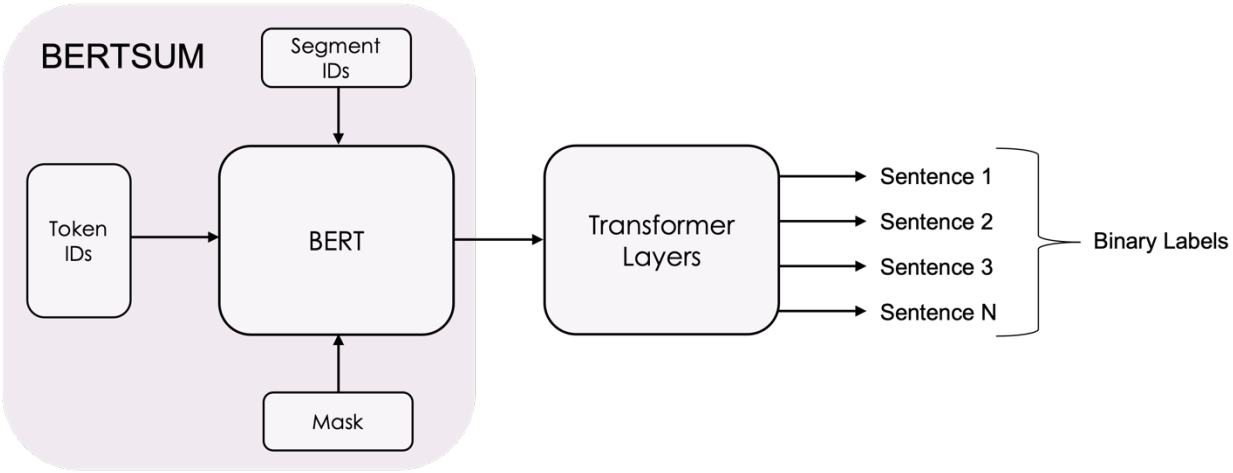


Figure 1. Extractive BERTSUM

To evaluate the summaries for a new document, the sentence scores obtained from the model is first ranked from highest to lowest, and only the top-3 sentences are selected as the summary. During sentence selection, the same *Trigram Blocking* employed by Liu et al. [9] was used to reduce redundancy in candidate sentences. The idea is to minimize the similarity between candidate sentences by excluding sentences that have trigram overlaps with the already selected candidate sentences.

b) Abstractive Summarization

For abstractive summarization, a sequence-to-sequence encoder-decoder architecture was implemented for abstractive BERTSUM with attentional RNNs. BERTSUM is used as the encoder and Recurrent Neural Network is used as the decoder with neural attention. For the neural attention in the decoder, two different implementations were considered: Bahdanau attention from TensorFlow sample tutorial and, TensorFlow add-ons API. During training, the preprocessed token ids, segment ids and token masks are fed into the encoder. The outputs (last hidden state) from the BERTSUM encoder are then fed into the attentional RNN decoder. The inputs to the decoder would be all the target token ids except the last token — [END] — and the outputs of the decoder would be target token ids shifted one timestep ahead so that the model can learn to predict the next target tokens at each timestep. Each epoch takes around 1.5 hours using the TensorFlow sample tutorial and around 2.5 hours using the TensorFlow add-ons API.

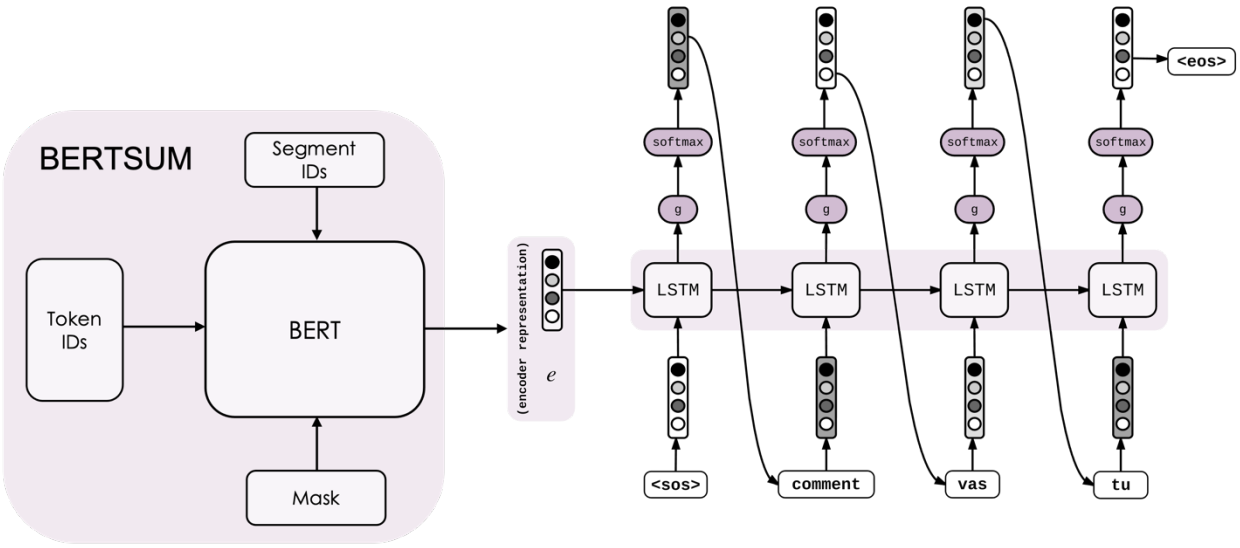


Figure 2. Abstractive BERTSUM-RNN

During inference, the [START] token is fed as initial input to the decoder to kickstart the summary generation. At each timestep, the decoder's output is combined with a weighted sum over the encoded input, to predict the next word. For generating abstractive summary, greedy search and beam search decoding were tried out. Greedy search decoding basically just selects the next highest probability token based on the decoder outputs, which is a SoftMax over the BERT vocabulary at any given timestep based on what it has seen so far. Beam search decoding, which is controlled by a beam width is more computationally expensive because it tries to select better sequence of tokens by searching alternative paths.

4 Results

Model	R1	R2	RL
Oracle	52.59	31.24	48.87
Oracle (pyrouge)	50.81	28.73	47.09
Lead-3	40.42	17.62	36.67
BERTSUM from Liu et al. [9]	43.05	20.16	39.45
BERTSUM with Keras	40.03	17.38	36.62

Table 2. Comparison of Extractive BERTSUM Models

Table 2 shows the performance comparison of extractive BERTSUM models on the CNN/DailyMail dataset with their F1 scores on the ROUGE-1, ROUGE-2, and ROUGE-RL metric. To compare the extractive BERTSUM models, scores are also provided for theoretical

models. For instance, the scores achieved by Oracle model are the upper bound as it predicts all the correct sentence labels. Likewise, Lead-3 selects the first three sentences in a document as its prediction.

When comparing the results between BERTSUM from Liu et al. and BERTSUM with Keras, the implementation in this paper is clearly few points below the original one. However, this can be explained with the Oracle (pyrouge) results. As can be seen in Table 2, there is a noticeable discrepancy between the Oracle scores reported by Liu et al. and the Oracle scores calculated by the pyrouge library. Since the ROUGE scores for BERTSUM with Keras are calculated by the same pyrouge library, the results are off by almost the same amount. Moreover, there are Machine Learning framework differences between Keras and PyTorch as well as optimization tricks employed by Liu et al. that might have contributed to the performance difference between the extractive BERTSUM models.

Figure 3 shows the progression of training loss for extractive BERTSUM as the model sees more batches of data across different epochs. It is clear from the figures that the model converges within the first epoch and training further with more epochs do not improve the model significantly. This is also confirmed by the training and validation loss figure as it shows that the validation loss starts going up after 2 epochs.

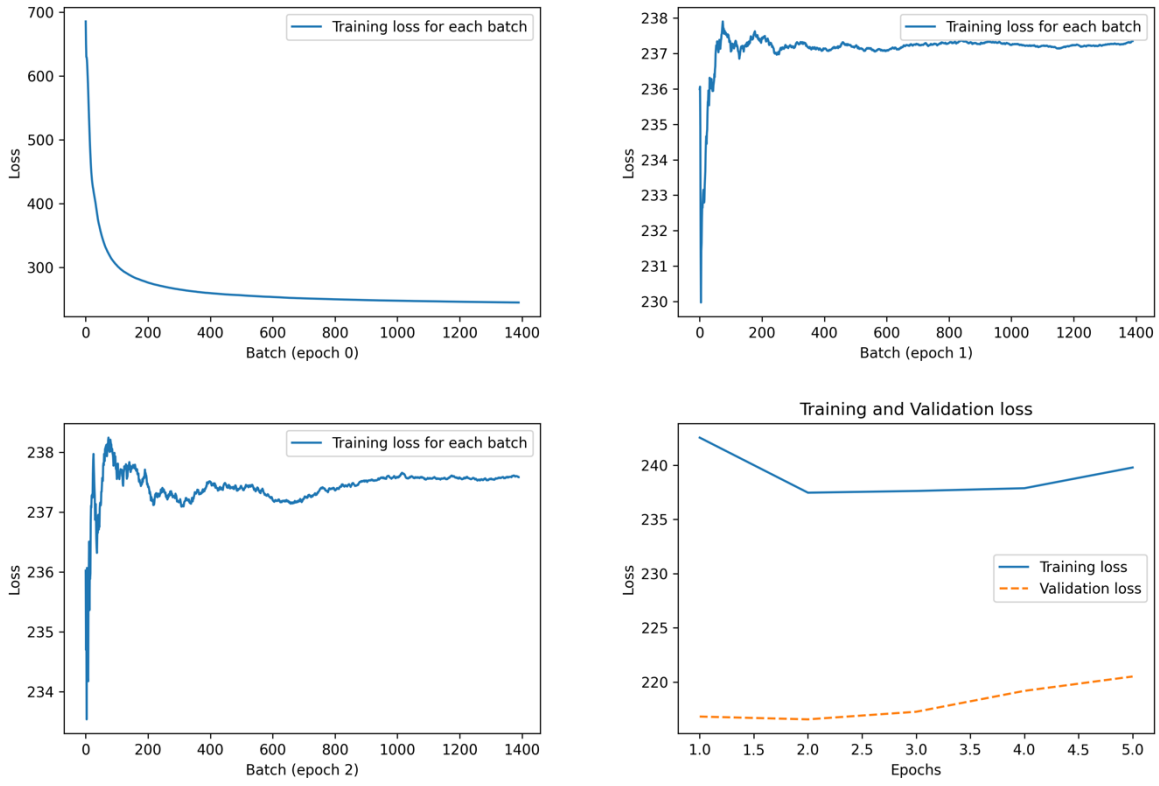
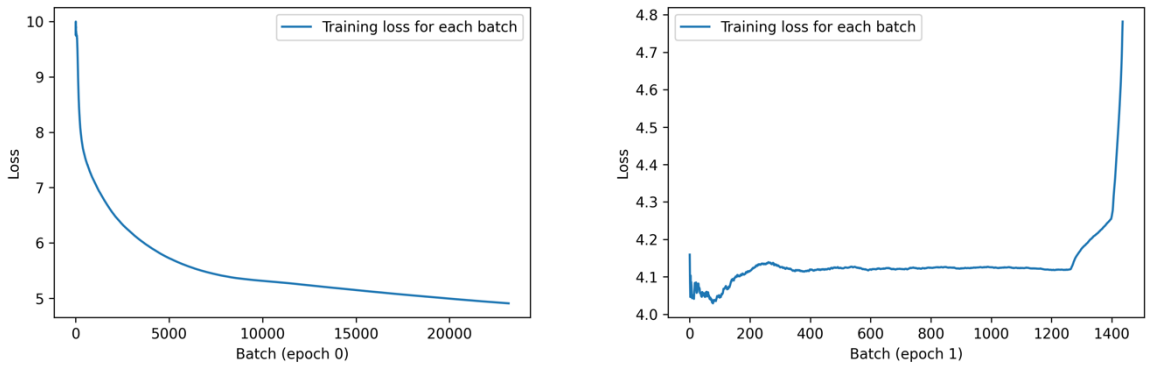
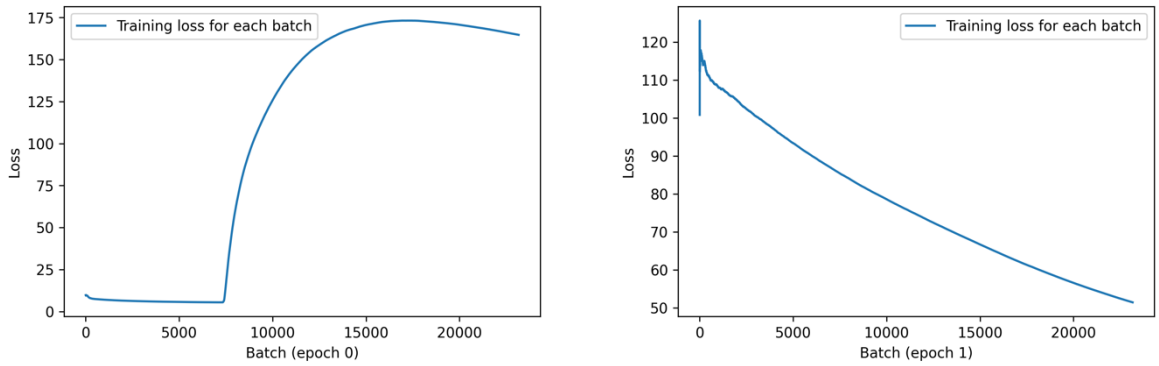


Figure 3. Extractive BERTSUM Training

As for abstractive BERTSUM-RNN, Figure 4 shows the progression of training loss for abstractive BERTSUM-RNN for two different hidden state initializations of RNN: (a) zero initialized and (b) pooler outputs from the BERTSUM encoder. The figures show the unstable training across epochs when using the TensorFlow add-ons API. Figure 5 shows the stable training across epochs when using the TensorFlow sample tutorial. The ROUGE scores for abstractive BERTSUM-RNN could not be calculated because the batch decoding (inference) encountered out-of-memory issues. The greedy search decoding done on the model with neural attention from TensorFlow sample tutorial produced garbage summary sentences. However, greedy search decoding done on the model with neural attention from TensorFlow add-ons API produced results that had decent grammar with appropriate vocabularies. Figure 6 lists some of the sample summaries.



(a) Initial Hidden State: Zero Initialized



(b) Initial Hidden State: Pooler outputs from the BERTSUM Encoder

Figure 4. Training Comparison for Neural Attention using TensorFlow Add-ons

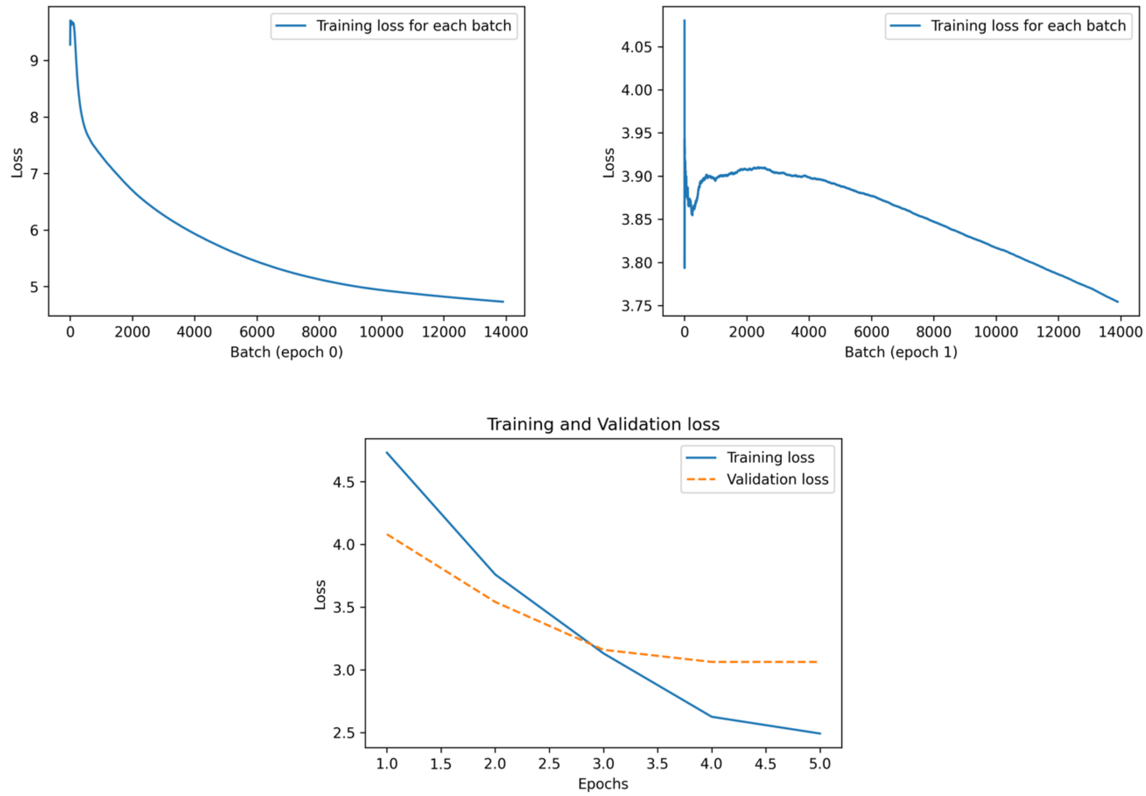


Figure 5. Training Comparison for Neural Attention using TensorFlow sample tutorial

- *the skeleton of the bones had been found out of the ancient ancient species of the ancient. the skeleton was found in the region of the region of the region of the region. the skeleton was found in the region of the region of the*
- *trey moses, who was ` the top of the prom ', ` she can't wait to prom. the song was ` the'of the club, the girl, the girl, says she was*
- *lihi yona filmed the dog in the air in brooklyn. the dog was filmed by the dog in the air. the dog was filmed by the dog and the dog*
- *sportsmail will receive nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike nike joe hart. the winner will receive nike nike nike nike nike nike nike nike nike nike nike. the winner will receive nike nike nike nike nike*
- *luis suarez scored twice as barcelona's first of the world's top of the club. luis suarez scored twice as barcelona's top of the club's top of the match. luis suarez scored twice as he crossed the club's 2 - 1 win*

Figure 6. Sample Generated (Candidate) Summaries

5 Conclusion

In this paper, extractive and abstractive text summarization models based on BERT pretrained language model were trained and evaluated on the CNN/DailyMail dataset. It was discovered that extractive BERTSUM with Keras do not achieve the same ROUGE scores as Liu et al. This is most likely due to the ROUGE library differences, ML framework differences (TensorFlow & PyTorch) and loss optimization tricks employed by the author. For abstractive BERTSUM-RNN, two different implementations of neural attention were considered. Training loss was unstable across epochs when using the TensorFlow add-ons API. However, training loss was stable across epochs when using the TensorFlow sample tutorial. The abstractive summarization results with greedy search were decent, which could be further improved with additional loss and memory optimization.

6 References

1. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed rep- resentations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.
2. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
3. J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
4. M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations" *arXiv preprint arXiv:1802.05365*, 2018.
5. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
6. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.
7. K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend," *Advances in neural information processing systems*, vol. 28, pp. 1693–1701, 2015.
8. C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.
9. Y. Liu and M. Lapata, "Text summarization with pretrained encoders," *arXiv preprint arXiv:1908.08345*, 2019
10. C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.