KEYWORD EXTRACTION IN BERT-BASED MODELS FOR REVIEWER SYSTEM

By

Joshua Newburn

Senior Thesis in Computer Engineering

University of Illinois Urbana-Champaign

Advisor: Dr. Wen-Mei Hwu

May 2023

Abstract

For keyword extraction tasks to obtain high quality phrases that encapsulate the overall meaning of a sentence, paragraph, or document, shallow systems have been employed to obtain satisfactory results. In this project, we research and implement methods to use deep learning methods to improve upon these systems for keyword extraction. We find that a pretrained BERT-model passed through the KeyBERT keyword extractor is effective at obtaining keywords from any given document, and we improve upon this result by fine-tuning a BERT-based model using token classification from pretrained bert-base-uncased and Adam optimization. We generate a larger quantity of high-quality phrases than created by the shallow system algorithm, and these keywords give better insight into the context of a paper within a specific domain gave by providing more descriptive keywords.

Subject Keywords: keyword extraction; deep learning; natural language processing; BERT

Contents

1. Introduction

A major problem in the research community is how to best match grad papers with potential reviewers. Reviewers are often given papers that they have no interest in reading, and this can be frustrating for reviewers, who want to review a paper that is well-matched for them.

One of the first tasks in matching a reviewer with a paper is to get an idea of what the material presented in the work is about. Reading through an entire grad paper to understand its context takes too much time, so authors are required to submit an abstract, which contains phrases that can give details about the paper's subject matter. These phrases, when properly selected, encapsulate general context about the paper and can be utilized to match a reviewer with the paper.

To extract keywords, simple methods can use POS tags to estimate the importance of some words. In such algorithms, parts of speech such as articles and prepositions are omitted, and verbs, nouns, and adjectives are usually chosen. Simple algorithms may have a word bank and select keywords that are found in that list. Keyword extraction using these shallow systems vary in accuracy and fail on unseen subjects where word banks are not readily available.

Out of this need for an automated paper reviewer selection to avoid manual selection and improve upon existing extraction methods, we implement a deep system by training BERT-based models. Deep systems allow the algorithm to have a comprehension of the overall meaning of the input document and output keywords that summarize the general context of the grad paper.

In the upcoming chapters, we present our findings using deep models to train a BERT-based model for keyword extraction. Chapter 2 is comprised of research context including information about BERT, LSTM RNNs, and KeyBERT. Chapter 3 is a description of research results, where we outline the training data, training procedure, testing methods, and findings. Finally, in chapter 4, we conclude with a summary of the major discoveries and potential paths for future exploration.

2. Research Context

2.1 Bidirectional Encoder Representations from Transformers

An industry standard for language comprehension in machine learning is Bidirectional Encoder Representations from Transformers (BERT). It is a Transformer stack developed by Google and trained on Wikipedia and Book Corpus through next-sentence prediction and word masking. Long-Short Term Memory (LSTM) Recursive Neural Networks (RNNs) can delete and add information, controlled by gates [1]. Transformers build upon the structure of LSTM RNNs through a unique bidirectional approach. [2]

Like the LSTM RNN structure, transformers encode relevant information for each word into a higher-dimension vector. Transformers, unlike RNNs, can go in both ways, gaining context of each word from previous words and words that come after it. This allows the model to gain a much better understanding of the overall language. [3]

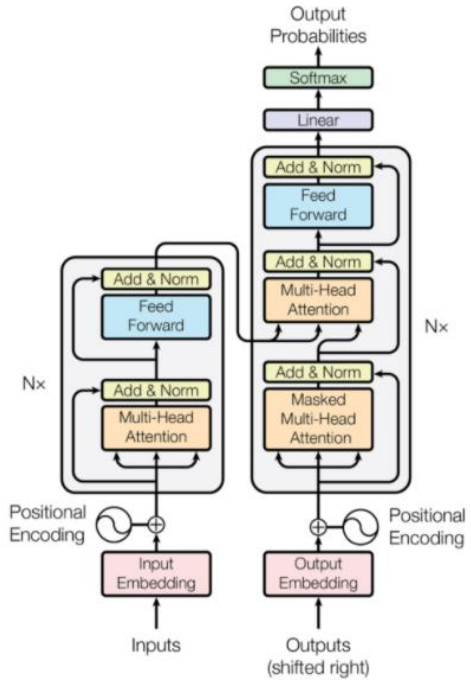

Figure 1 – a visualization of the BERT architecture that shows its bidirectional, multi-head attention mechanism for deep language comprehension. [4]

2.2 KeyBERT

To develop effective keyword extractor, we use Maarten Grootendorst's KeyBERT, which employs a BERT-based model in deep language comprehension. KeyBERT creates a list of candidate key phrases of length n, which will be evaluated, and the best phrases are outputted. Both the document and candidates are embedded into higher-dimension vectors and evaluated using cosine similarity. The candidates with the highest cosine similarity score are delivered as the best keywords [5].

KeyBERT is effective for broad tasks but falls short when it comes to accuracy in specific domains. Using default models, KEYBERT extracts phrases on what is considered important in the context of all the English language. However, most research papers fall within certain sub-categories.

Sub-catagories have their own criteria for what is considered important. For example, a default KEYBERT model might consider "scientific hypothesis" to be an important in a physics abstract, since that word is not often found in other papers in the context of all written papers. However, in the domain of physics, that is a very common phrase and doesn't uncover much about the contents of the paper. A default model falls short since it has not been fine tuned for a specific domain.

3. Description of Research Results

Our training data comprised of pairs of abstracts and their corresponding keyword phrases. These phrases were generated using a simple algorithm, which employed shallow techniques such as POS tagging and word bank matching. This dataset included a set of keywords for each abstract.

We use a model training method specific to keyword extraction from Ishaan Batra. [6] To train the model, we take in test data in two text files. One is a document containing the paragraphs of information to be summarized. The second is a list of keywords designated for that document, with one phrase on each line. This testing data goes through a text conversion, including deleting punctuation. The modified text is then tokenized using BertForTokenClassification from pretrained bert-base-uncased and sent through Adam Optimizer for fine-tuning. The hyperparameters to be specified include learning rate, number of epochs, batch size, and sequence length.

We format our training data to match what is expected and used a training set of 100 abstracts. We train our data by specifying epoch number, learning rate, batch size, and sequence length, to obtain a fine-tuned model that we could test through KeyBERT.

We test the model by sending abstracts through KeyBERT using our fine-tuned model and using the default pretrained model. Looking for matches in our data with the dataset, we count the number of keywords in the output that are found in the testing data. For each model, they are constrained to output the same number of keywords with identical settings and hyperparameters. For both models, we specify the number of keywords to be within 1 and 3 and that there are no stop words (a, the, is, are, etc.)

```
default_kw_model = KeyBERT(model= “”)
kw_model = KeyBERT(model= “model.pt”)
```

Figure 2 – a code snippet from the research model specification. The two shown here are the default model and “model.pt” a model trained using 100 abstracts and their keywords

Table 1 - Accuracy vs Model Used

Model	Training method	Size of training data (# of abstracts)	Size of testing data (# of abstracts)	Accuracy (# of matched keywords)
Default	Pretrained only	100	20	47
“Model.pt”	Pretrained + fine-tuning	100	20	50

As seen in Table 1, we discover that performance of a default model from a test size of 20 abstract is able to have 47 matches with keywords found both in the test data and outputted results. After fine-tuning a model named “Model.pt,” its performance increased to 50 matched keywords, an improvement of 6%.

Results were further improved by considering the increasing the diversity of our keyword output. We specify a diversity parameter for the KeyBERT extraction (Figure 1).

```
kw_model.extract_keywords(abstract, diversity=0.2):
kw_model.extract_keywords(abstract):
kw_model.extract_keywords(abstract, diversity=0.7):
```

Figure 3 – a code snippet from the research keyword extraction function. The different diversity settings are 0.2, default (about 0.4), and 0.7. The effects of the increasing the diversity settings are in Table 2

Table 2 - Accuracy vs Diversity

Diversity	Model name	Size of training data (# of abstracts)	Size of testing data (# of abstracts)	Accuracy (# of matched keywords)
0.2	Model.pt	100	20	36
Default (~0.4)	Model.pt	100	20	50
0.7	Model.pt	100	20	77

Higher diversity meant the outputs were giving fewer duplicate keywords and more chance to have a wider range of possibilities (Table 2). By combining a pretrained BERT-model and training it on the training data generated by the simple algorithm, we are able to see better results, and in many cases, an improvement upon the simple algorithm. In some cases, the simple algorithm outputs one or two keywords. Our model generates several other high-quality phrases to complement the ones generated by the simple algorithm.

4. Conclusion

We find that training a BERT-based model in a developed keyword extractor brings the capability of methods that surpass simplistic algorithms in their keyword extraction. The simple algorithm's inability to generate numerous keywords in some cases is likely due to the lack of words in the word bank for that specific abstract.

A key insight into the the training was the data employed to train the model. This data was created by a simplistic algorithm, and thus the quality of training was only mediocre. Improvement of training data would lead to better training and performance. This data could be extracted by a few professions or experts in the domain after reading the abstract, or better yet, the entire research paper. Alternatives could also be explored that require less training data.

Further exploration can be done in other methods. SpanBERT shows promise to represent spans of text by extending BERT in its masking system to mask tokens instead of just words [7]. NSP-BERT trains at the token-level [8], and DocBERT which aims at document classification [9]. Other methods that may be worth pursuing are Named Entity Recognition such as Bert-Base-NER to recognize types of entities [10], Prompt-Based learning such as NSP-BERT to train a model using a question-answer system [11], and relation extraction to classify documents into their corresponding categories names from the training set.

These and other methods can improve the matching of grad paper to reviewers and reduce the time and accuracy of the selection process. These deep systems show great promise to improve the efficiency of many systems that previously relied on shallow methods and can bring about improvements through deep language comprehension.

References

[1] "Understanding LSTM networks," *Understanding LSTM Networks -- colah's blog*. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 11-Nov-2022].

[2] Maxime, "What is a Transformer?," *Medium*, 05-Mar-2020. [Online]. Available: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04. [Accessed: 11-Nov-2022].

[3] P. D. Michel Kana, "Bert for dummies - step by step tutorial," *Medium*, 15-Jun-2020. [Online]. Available: https://towardsdatascience.com/bert-for-dummies-step-by-step-tutorial-fb90890ffe03. [Accessed: 13-Nov-2022].

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv.org*, 06-Dec-2017. [Online]. Available: https://arxiv.org/abs/1706.03762. [Accessed: 13-Nov-2022].

[5] M. Grootendorst, "Keyword extraction with bert," *Maarten Grootendorst*, 28-Oct-2020. [Online]. Available: https://www.maartengrootendorst.com/blog/keybert/. [Accessed: 15-Dec-2022].

[6] I. Batra, "Bert-keyword-extractor: Deep keyphrase extraction using bert," *GitHub*. [Online]. Available: https://github.com/ibatra/BERT-Keyword-Extractor. [Accessed: 29-Dec-2022].

[7] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "Spanbert: Improving pre-training by representing and predicting spans," *arXiv.org*, 18-Jan-2020. [Online]. Available: https://arxiv.org/abs/1907.10529. [Accessed: 05-Jan-2023].

[8] S. W. Tee, “Bert-relation-Extraction: Pytorch implementation,” *GitHub*. [Online]. Available: https://github.com/plkmo/BERT-Relation-Extraction. [Accessed: 05-Jan-2023].

[9] A. Adhikari, A. Ram, R. Tang, and J. Lin, “DocBERT: Bert for Document Classification,” *arXiv.org*, 22-Aug-2019. [Online]. Available: https://arxiv.org/abs/1904.08398. [Accessed: 05-Jan-2023].

[10] D. S. Lim, “bert-base-ner,” *Hugging Face*. [Online]. Available: https://huggingface.co/dslim/bert-base-NER. [Accessed: 05-Jan-2023].

[11] Y. Sun, Y. Zheng, C. Hao, and H. Qiu, “NSP-Bert: A prompt-based few-shot learner through an original pre-training task--next sentence prediction,” *arXiv.org*, 18-Oct-2022. [Online]. Available: https://arxiv.org/abs/2109.03564. [Accessed: 05-Jan-2023].